▼ AMSTRAD ▼ SPECTRUM ▼ COMMODORE ▼ IBM ▼ MSX ▼

Informática y programación

13

PASO A PASO



PROGRAMAS EDUCATIVOS
PROGRAMAS DE UTILIDAD
PROGRAMAS DE GESTION
PROGRAMAS DE JUEGOS

▼ BASIC ▼ MAQUINA ▼ PASCAL ▼ LOGO ▼ OTROS LENGUAJES ▼

▼ TECNICAS DE ANALISIS Y DE PROGRAMACION ▼

▼ EDICIONES ▼ SIGLO ▼ CULTURAL ▼

Informática y programación 13

PASO A PASO



PROGRAMAS EDUCATIVOS
PROGRAMAS DE UTILIDAD
PROGRAMAS DE GESTION
PROGRAMAS DE JUEGOS

▼ BASIC ▼ MAQUINA ▼ PASCAL ▼ LOGO ▼ OTROS LENGUAJES ▼
▼ TECNICAS DE ANALISIS Y DE PROGRAMACION ▼

▼ EDICIONES ▼ SIGLO ▼ CULTURAL ▼

Una publicación de

EDICIONES SIGLO CULTURAL, S.A.

Director-editor:
RICARDO ESPAÑOL CRESPO.
Gerente:
ANTONIO G. CUERPO.
Directora de producción:
MARIA LUISA SUAREZ PEREZ.
Directores de la colección:
MANUEL ALFONSECA, Doctor Ingeniero de Telecomunicación y Licenciado en Informática.
JOSE ARTECHE, Ingeniero de Telecomunicación.
Diseño y maquetación:
BRAVO-LOFISH.
Fotografía:
EQUIPO GALATA.
Dibujos:
JOSE OCHOA

TECNICAS DE PROGRAMACION: Manuel Alfonseca, Doctor Ingeniero de Telecomunicación y Licenciado en Informática. TECNICAS DE ANALISIS: José Arteche, Ingeniero en Telecomunicación. LENGUAJE MAQUINA 8086: Juan Rojas Licenciado en Ciencias Físicas e Ingeniero Industrial. PASCAL: Juan Ignacio Puyol, Ingeniero Industrial. PROGRAMAS (educativos, de utilidad, de gestión y de juegos): Francisco Morales, Técnico en Informática y colaboradores. Coordinador de AULA DE INFORMATICA APLICADA (AIA): Alejandro Marcos, Licenciado en Ciencias Químicas. BASIC: Esther Maldonado, Diplomada en Arquitectura. INFORMATICA BASICA: Virginia Muñoz, Diplomada en Informática. LENGUAJE MAQUINA Z-80: Joaquín Salvachúa, Diplomado en Telecomuniación y José Luis Tojo, Diplomado en Telecomunicación. LENGUAJE MAQUINA 6502: (desde el tomo 5): Juan José Gómez, Licenciado en Química. LOGO: Cristina Manzanero, Licenciada en Informática. APLICACIONES: Soledad Tamariz, Diplomada en Telecomunicación. OTROS LENGUAJES (sistemas operativos): Domingo Villaseñor, Diplomado en Informática, y Lenguaje C: Enrique Serrano, Ingeniero en Telecomunicación.

Ediciones Siglo Cultural, S.A.
Dirección, redacción y administración:
Pedro Teixeira, 8, 2.ª planta. Teléf. 810 52 13. 28020 Madrid.
Publicidad:
Gofar Publicidad, S.A. Benito de Castro, 12 bis. 28028 Madrid.
Distribución en España:
COEDIS, S.A. Valencia, 245. Teléf. 215 70 97. 08007 Barcelona.
Delegación en Madrid: Serrano, 165. Teléf. 411 11 48.
Distribución en Ecuador: Muñoz Hnos.
Distribución en Perú: DISELPESA.
Distribución en Chile: Alfa Ltda.
Importador exclusivo Cono Sur:
CADE, S.R.L. Pasaje Sud América, 1532. Teléf.: 21 24 64.
Buenos Aires - 1.290. Argentina.

ISBN del tomo: 84-7688-105-3
ISBN de la obra: 84-7688-068-7
Fotocomposición:
ARTECOMP, S.A. Albarracín, 50. 28037 Madrid.
Imprime:
MATEU CROMO. Pinto (Madrid).

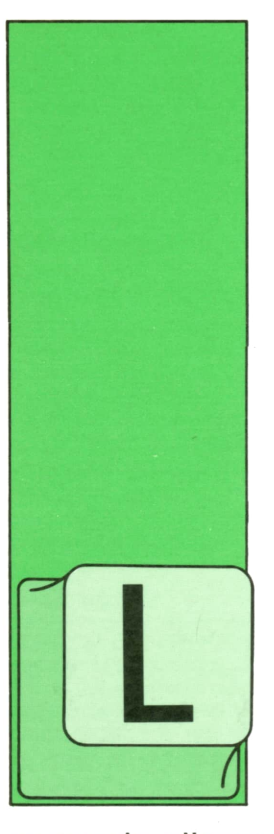
Depósito legal: M-5-677-1987
Printed in Spain - Impreso en España.
Suscripciones y números atrasados:
Ediciones Siglo Cultural, S.A.
Pedro Teixeira, 8, 2.ª planta. Teléf. 810 52 13. 28020 Madrid.
Mayo, 1987.
P.V.P. Canarias: 335,–.

INDICE

BASIC



La instrucción GOTO



LAS instrucciones de los programas que hemos visto hasta ahora se ejecutaban siempre según el orden creciente del número de línea. Sin embargo, este orden se puede alterar utilizando la instrucción GOTO, que posibilita la transferencia del control de ejecución a cualquier otra línea del programa, anterior o posterior. El formato general es el siguiente:

GOTO <número de línea>

La instrucción GOTO realiza un salto incondicional al número de línea indicado.

El programa 1 es un ejemplo del funcionamiento de GOTO.

```
10 REM **************************
20 REM *  CASTIGO AL ORDEÑADOR  *
30 REM **************************
40 CLS
50 PRINT "VACA SE ESCRIBE CON V"
60 GOTO 50
```

En el programa hemos formado lo que se denomina un bucle infinito, es decir, un conjunto de instrucciones que se repite infinitas veces. En este caso el bucle infinito se está formando por las líneas 50 y 60. Esto hace que la cadena VACA SE ESCRIBE CON V se vaya imprimiendo en cada una de las líneas de la pantalla. Cuando la pantalla se complete, las frases irán desapareciendo por la parte superior para poder seguir imprimiendo en las líneas inferiores indefinidamente.

Evidentemente, cuando un programa tiene un bucle infinito, la ejecución no se detiene de forma espontánea. Para poder detener la ejecución de un bucle de este tipo normalmente tendremos que pulsar una o dos teclas, dependiendo del tipo de ordenador. En la figura 1 podemos ver las teclas usadas en las máquinas en estudio.

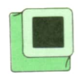
AMSTRAD	ESC
COMMODORE	RUN/STOP
IBM	CTRL + BREAK
MSX	CTRL + STOP
SPECTRUM	BREAK



Teclas que detienen la ejecución de un bucle infinito.

Por lo general, un bucle infinito no suele tener mucho sentido; por tanto, deben evitarse en los programas, puesto que no conducen a nada. Sin embargo, pueden utilizarse en programas muy especiales, como, por ejemplo, la transformación del teclado del ordenador en un teclado musical.

Esto no significa que la instrucción GOTO sea de poca utilidad, todo lo contrario, podemos usarla en gran cantidad de programas, como veremos a continuación.



Los bucles condicionales

Podemos combinar la instrucción GOTO con la instrucción IF-THEN de modo

que consigamos saltos condicionales, es decir, los bucles formados ya no serán infinitos, sino que dependerán de una condición. Esto dará gran versatilidad a nuestros programas.

Analicemos la estructura de la figura 2, que formará parte de muchos programas.

```
10  LET C=0
20  LET C=C+1
        .
        .
        .
        .
60  IF C<5 THEN GOTO 20
```


Esqueleto básico de un bucle condicional.

En la línea 10 se asigna un valor inicial a la variable C que va a actuar como contador del número de veces que se va a repetir el bucle. Si el valor inicial es 0, como en este caso, se puede prescindir de la línea en todos los ordenadores, excepto en el SPECTRUM.

En la línea 20 se efectúa el incremento del contador. En este caso se trata de contar de uno en uno; por tanto, se suma 1 al contenido de la variable C y el resultado se vuelve a almacenar en C, borrándose el antiguo contenido. Por ello la instrucción de la línea 20 podría traducirse como «Guarda en la variable C lo que había antes más 1». Lógicamente, si interesara un incremento distinto de 1 se podría utilizar también incluso valores negativos, que darían lugar a decrementos. Por ejemplo:

```
20 LET C=C+5
```

contaría de cinco en cinco y

```
20 LET C=C−1
```

contaría de uno en uno, pero hacia atrás.

Finalmente, todas las líneas comprendidas entre la 20 y la 60 constituirán el cuerpo del bucle y se repetirán tantas veces como indique la línea 60, donde se encuentra la condición de finalización del bucle: sólo si el valor de C es menor que 5, el control volverá a la línea 20. Si no se cumple esta condición, la ejecución del programa pasará a la línea siguiente, si existe, y si no, se detendrá definitivamente.

En el programa podemos ver un ejemplo concreto:

```
10 REM **************************
20 REM *  EJEMPLO DE CONTADOR   *
30 REM **************************
40 CLS
50 LET C=0
60 LET C=C+1
70 PRINT "ESTE MENSAJE SE REPITE 15 VECES"
80 IF C<15 THEN GOTO 60
```

Al ejecutarlo podemos observar que la cadena de la línea 70 se imprime en pantalla 15 veces, tal y como está establecido en la condición de la línea 80.

Veamos otro ejemplo más práctico. El programa 3 utiliza un bucle para imprimir en pantalla los números naturales, sus cuadrados y sus cubos del intervalo que deseemos.

```
10 REM ********************
20 REM *  TABLA NUMERICA  *
30 REM ********************
40 CLS
50 INPUT "TECLEE EL NUMERO INICIAL ";NI
60 INPUT "TECLEE EL NUMERO FINAL ";NF
70 CLS
80 IF NI>NF THEN PRINT "TECLEE PRIMERO
   EL MENOR":GOTO 50
90 PRINT "NUMERO";TAB(12);"CUADRADO";
   TAB(24);"CUBO"
100 PRINT "______";TAB(12);"________";
    TAB(24);"____"
110 PRINT
120 LET C=NI
130 PRINT C;TAB(12);C*C;TAB(24);C^3
140 LET C=C+1
150 IF C<=NF THEN GOTO 130
```

La condición de la línea 80 no permite que el número inicial sea mayor que el final; por tanto, si tecleamos los números al contrario aparecerá en pantalla un mensaje de aviso y volverá a pedirnos los dos números. Las líneas 90 y 100 se encargan de imprimir en la parte superior de la pantalla un encabezamiento correcto. En la línea 120 se asigna el valor inicial al contador, que, lógicamente, coincide con el valor inicial que hemos tecleado. Finalmente las líneas 130 a 150 constituyen el bucle que se repetirá hasta que el contador alcance el número final tecleado. En este caso hemos invertido el orden de las líneas, es decir, primero se realiza la impresión en pantalla y después se incrementa el contador.

En la figura 3 podemos ver la presentación en pantalla tras una posible ejecución.

Número	Cuadrado	Cubo
5	25	125
6	36	216
7	49	343
8	64	512
9	81	729
10	100	1.000
11	121	1.331
12	144	1.728
13	169	2.197
14	196	2.744
15	225	3.375
16	256	4.096
17	289	4.913
18	324	5.832
19	361	6.859
20	400	8.000
OK		


Presentación en pantalla del programa 3.

En la figura 4 se resumen las características generales de estos bucles condicionales.

1. Su utilización permite realizar de forma repetida una serie de acciones.
2. Son estructuras perfectamente delimitadas:
   - Cuentan siempre con una primera orden que señala su comienzo.
   - Y con otra, que se puede denominar de cierre, que señala el final del bucle.
3. La forma más sencilla de un bucle con IF-THEN responde al esquema:
   Aumento del contador: Instrucción de comienzo.
   IF-THEN: Instrucción de cierre: Se comprueba el contador.
4. Todas las instrucciones incluidas entre las de comienzo y cierre serán las que se ejecuten repetidas veces.
5. El número de veces que se realice el bucle se especifica en la condición final IF-THEN, donde se comprueba si el contador ha alcanzado ya el valor máximo que limita el número de vueltas que se darían.


Características generales de los bucles condicionales.

Vamos a ver ahora un ejemplo algo diferente. El programa 4 nos permite hallar la media aritmética de una serie de datos numéricos. En este caso concreto calcula la edad media de un conjunto de personas. Este programa podríamos desarrollarlo con un contador y un bucle condicional si conociéramos de antemano el número de datos que vamos a introducir. Sin embargo, vamos a suponer que no conocemos este número; por tanto, utilizaremos un bucle del que sólo podremos salir al teclear un dato ficticio, 1. De todas formas, usaremos un contador para saber cuántos datos hemos introducido.

```
10 REM *******************
20 REM *  MEDIA DE DATOS  *
30 REM *******************
40 CLS
50 LET C=0:LET S=0
60 INPUT "TECLEA LA EDAD (-1 PARA TERMI
   NAR) ";E
70 CLS
80 IF E=-1 THEN GOTO 130
90 IF E<=0 OR E>100 THEN PRINT "EDAD NO
   VALIDA":GOTO 60
100 LET C=C+1
110 LET S=S+E
120 GOTO 60
130 LET M=S/C
140 PRINT "NUMERO DE DATOS: ";TAB(20);C
150 PRINT :PRINT
160 PRINT "MEDIA ARITMETICA: ";TAB(20);M
```

En la línea 50 asignamos el valor 0 a dos variables: C y S. La variable C actuará como contador, mientras que la variable S funcionará como un **acumulador,** es decir, en esta variable se va a realizar la suma de todos los datos que tecleemos. La condición de la línea 80 comprueba si hemos tecleado el dato ficticio −1, en cuyo caso la ejecución saldrá del bucle, transfiriéndose el control a la línea 120, donde se calcula la media para, a continuación, imprimir en pantalla los resultados (líneas 140 a 160). La condición de la línea 90 sirve para comprobar que no hemos tecleado una edad absurda (es imposible tener menos de 0 años y muy difícil superar los 100). En la línea 100 se incrementa el contador y en la 110 se suma la edad tecleada al acumulador; de este modo al salir del bucle tendremos todas las edades sumadas en la variable S, y podremos dividirla por C (número de datos) para calcular la media.

En la figura 5 podemos ver el resultado de una posible ejecución.



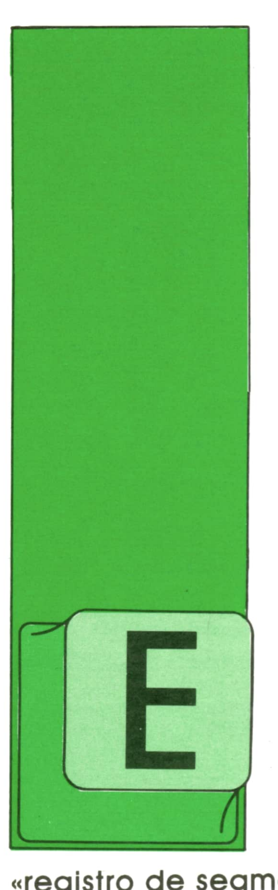


Presentación en pantalla del programa 4.

MAQUINA 8088



Referencias explícitas a la memoria



EN el tomo 4 se trató de las referencias a la memoria y se dijo que la dirección física (20 bits) de una posición de memoria se obtienen sumando los contenidos de un «registro de segmento» (16 bits) y un «registro de desplazamiento» (16 bits), pero desplazando 4 bits a la izquierda el «registro de segmento» antes de la suma.

El 8088 accede a las áreas de código, datos origen, datos destino y pila de ejecución, utilizando implícitamente las parejas de registros CS:IP, DS:SI, ES:DI y SS:SP. Por esta razón, instrucciones como LODSW, STOSW y RET no llevan operandos en los que se especifique la memoria a utilizar, ya que por definición acceden a las áreas de datos origen, datos destino y pila, respectivamente, sin posibilidad de otra alternativa.

Independientemente de esto, en muchas instrucciones ejecutables uno de los operandos (pero nunca los dos) puede hacer referencia explícita a un byte o una palabra de memoria. En estas referencias, la dirección física de la memoria referenciada (20 bits) se obtiene por el mismo procedimiento a partir de un «registro de segmento» (16 bits) y un «desplazamiento» (16 bits). En cada sentencia se pueden especificar estos componentes, cosa que no ocurre en las referencias implícitas.

En resumen, en las instrucciones que hacen referencias explícitas a la memoria hay que definir:

— El «desplazamiento».

— El «registro de segmento» (opcional).

— La longitud de la unidad de memoria referenciada (opcional). Es decir, hay que definir si se trata de una palabra o de un byte.

El **desplazamiento** se puede definir utilizando constantes, etiquetas, registros base (BX o BP) y registros índice (SI o DI) combinados de muy variadas formas. A la hora de codificar las instrucciones se deben tener en cuenta las siguientes reglas sintácticas:

— Los registros deben ir entre corchetes (ejemplo: (SI)).

— Si se utilizan dos registros, ambos pueden ir en diferentes corchetes (ejemplo: (BX) (DI) o encerrarse en los mismos separados por el signo más (ejemplo (BX+DI)).

— Las etiquetas deben ir fuera de corchetes (ejemplo: LISTA).

— Las constantes deben ir entre corchetes si van solas (ejemplo (8)) y pueden ir fuera de corchetes si van acompañadas (ejemplo (BP)+8).

En las referencias explícitas a la memoria se puede definir el **registro de segmento** que se desee utilizar. Para ello basta con escribir uno de los cuatro posibles «registros de segmento» (DS, ES, CS y SS), delante del desplazamiento, separando ambos componentes por dos puntos. Ejemplos: CS:LISTA o ES:(BP+SI)+4.

En caso de no especificarse, se utiliza un «registro de segmento» elegido de acuerdo con las siguientes reglas:

1. Si en el desplazamiento intervienen los registros BX, SI o DI, se toma por omisión el registro DS.
2. Si en el desplazamiento interviene el registro BP, se toma por omisión el registro SS.

3. En caso de utilizar una etiqueta, se toma por omisión el registro que en la sentencia ASSUME se haya asociado al segmento al que pertenece la etiqueta.

Las reglas se aplican en el orden definido, de modo que, en caso de que puedan aplicarse dos o más reglas contradictorias, prevalecen las últimas sobre las primeras.

Por último, también se puede definir en la instrucción la **longitud** de la unidad de memoria referenciada. Para ello, hay que anteponer las palabras WORD PTR si se desea hacer referencia a una palabra y BYTE PTR si se desea hacer referencia a un byte. Ejemplos: WORD PTR LISTA o BYTE PTR CS:LISTA (SI)(BP)+4.

La especificación de longitud es obligatoria sólo en los casos en los que dicha longitud no se pueda deducir del contexto. Si la etiqueta LISTA se define como «palabra» (word), no es necesario usar WORD PTR en las referencias en las que aparezca dicha etiqueta.

Tipos de direccionamiento de la memoria

Las diferentes formas de referenciar explícitamente la memoria que acabamos de ver, dan lugar a los siguientes tipos de direccionamiento:

— Direccionamiento directo. En el que se especifica una etiqueta, seguida opcionalmente de una constante que se suma o resta a la misma. Por ejemplo, si se ha definido la etiqueta LISTA, se pueden escribir instrucciones que contengan como operando expresiones de los siguientes tipos:

(20)
LISTA
LISTA (20) o LISTA+20
LISTA (−7) o LISTA−7

— Direccionamiento indirecto por medio de un registro base. En el que se especifica el registro BX o el registro BP (que son los que pueden actuar como registros base) y una constante opcional aditiva o sustractiva. Ejemplos:

(BX)
(BP)+4
(BX)−32

Direccionamiento indirecto por medio de un registro índice. En el que se especifica el registro SI o el registro DI (que son los que pueden actuar como registros índices) y opcionalmente una constante aditiva o sustractiva. Ejemplos:

(SI)
(SI)+9
(DI)−17

Direccionamiento indirecto utilizando un registro base y un registro índice. En el que se especifica uno de los dos registros base y uno de los dos registros índice, seguidos opcionalmente por una constante aditiva o sustractiva. Ejemplos:

(SI+BP) o (SI) (BP)
(BX+SI)+9 o (BX) (SI)+9
(BP+DI)−17 o (BP) (DI)−17



La instrucción MOV

Esta es una de las sentencias básicas del ensamblador y por ello va a ser la primera sentencia ejecutable que vamos a describir con detalle. Mediante esta sentencia se puede copiar un byte o una palabra desde un «origen» a un «destino».

El formato de la sentencia MOV es el siguiente:

```
[etiqueta]   MOV   destino,origen
```

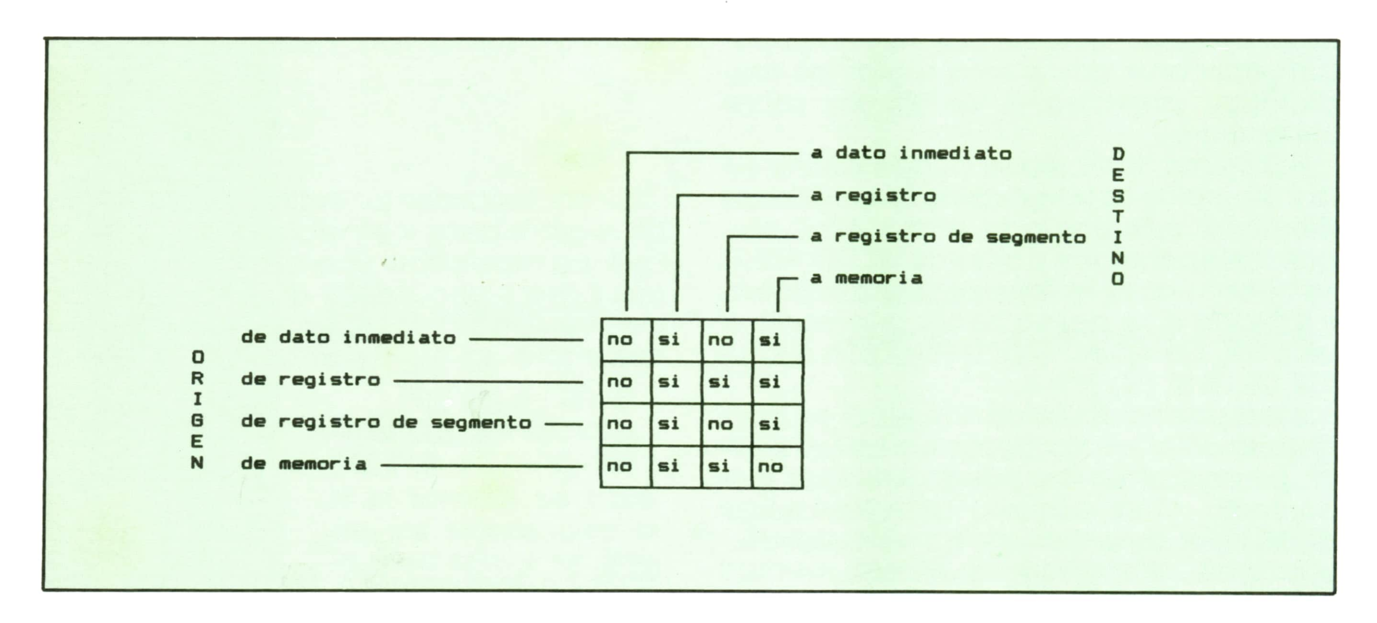
Donde la etiqueta es opcional como en todas las sentencias ejecutables. MOV es el código de operación. Y «origen» y «destino» son los dos operandos asociados a esta sentencia. Estos operandos pueden ser: datos inmediatos, registros, registros de segmento y referencias explícitas a la memoria.

Se denominan «datos inmediatos» u «operandos inmediatos» a las constantes que aparecen escritas directamente en la sentencia y que representan el valor que se quiere copiar al destino.

Todos los registros pueden ser «origen» y «destino» de la instrucción MOV (excepto el CS que no puede ser destino). A la hora de analizar las combinaciones posibles se separan los registros de segmento (DS, ES, SS y CS) de todos los de-

más, porque presentan algunas restricciones.

De las 16 combinaciones que pueden surgir cuando estos cuatro conceptos se utilicen como origen y como destino, hay nueve combinaciones válidas y siete inválidas que se resumen en la siguiente tabla:



A continuación se dan ejemplos de todos los tipos de instrucciones MOV, donde por simplicidad se ha utilizado la palabra MEMORIA para expresar cualquiera de las formas de referenciar explícitamente la memoria que se han visto al principio de este tema.

	a registro	a registro de segmento (1)	a memoria
de dato inmediato	MOV AX,12000 MOV CL,5	no no	MOV MEMORIA,1234 MOV MEMORIA,1
de registro de palabra AX,BX,CX,DX,SI,DI,BP,SP	MOV AX,BX MOV CX,DX	MOV ES,AX MOV SS,DX	MOV MEMORIA,BX MOV MEMORIA,CX
de registro de byte AH,AL,BH,BL,CH,CL,DH,DL	MOV AL,BL MOV CL,AH	no	MOV MEMORIA,BH MOV MEMORIA,AH
de registro de segmento DS,ES,SS,CS	MOV CX,DS MOV AX,ES	no	MOV MEMORIA,CS MOV MEMORIA,SS
de memoria	MOV DH,MEMORIA MOV BX,MEMORIA	MOV ES,MEMORIA MOV DS,MEMORIA	no no

(1) Excepción: El registro de segmento CS no se puede usar como "destino".

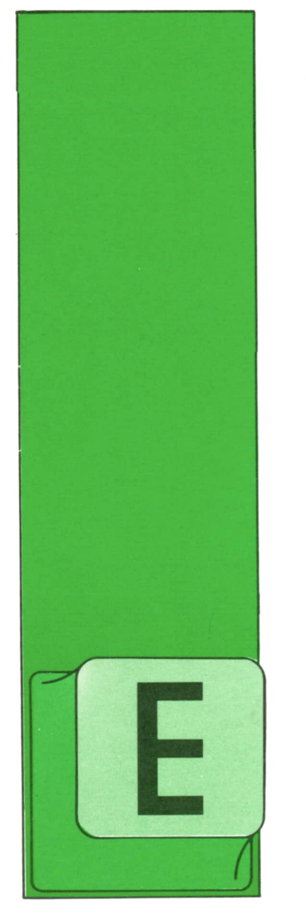
Se incluye, por último, el listado del programa PROG3, que copia un mensaje desde un área de datos al buffer de pantalla. El programa no pretende usar el método más eficaz de realizar esta tarea, sino utilizar la mayor variedad posible de ejemplos de instrucciones MOV.

En un comentario se recuerda que la dirección del buffer de pantalla debe cambiarse para que funcione con la pantalla de color.

```
            NAME     PROG3                    ; Programa PROG3
;---------------------------------
DATOS       SEGMENT                           ; Comienzo del segmento DATOS
;
MENSAJE     DB       'ABCDEFGHIJKLM'          ; Definición de la etiqueta MENSAJE
            DB       'NOPQRSTUVWXYZ$'         ; con datos literales.
;
DATOS       ENDS                              ; Fin del segmento DATOS
;---------------------------------
CODIGO      SEGMENT                           ; Comienzo del segmento CODIGO
NOMBRE1     PROC     FAR                      ; Comienzo del procedimiento NOMBRE1
;
            ASSUME   CS:CODIGO,ES:DATOS       ; Registros de segmentos asumidos.
;
            PUSH     DS                       ; Prepara en la pila
            MOV      AX,0000                  ; la dirección
            PUSH     AX                       ; de terminación.
;
            MOV      AX,DATOS                 ; Carga el registro de segmento ES con
            MOV      ES,AX                    ; la dirección del segmento de datos.
;
            MOV      AX,0B000H                ; 0B800H (Para pantalla de color)
            MOV      DS,AX                    ; Carga el registro de segmento DS con
                                              ; la dirección del buffer de pantalla.
;
            MOV      AH,0F1H                  ; Atributo del mensaje en la pantalla.
;
            MOV      BP,500H                  ; Posición inicial sobre la pantalla.
;
            MOV      BX,0                     ; Valor inicial del registrc que se
                                              ; utilizará para referenciar los
                                              ; cracteres del mensaje.
;
ETIQUETA1:  MOV      AL,ES:MENSAJE [BX]       ; Copia caracter del mensaje en AL
            INC      BX                       ; Incrementa el contador de caracteres
;
            CMP      AL,'$'                   ; Comprueba si es el último caracter
            JE       ETIQUETA2                ; de MENSAJE para salir del bucle.
;
            MOV      DS:[BP],AX               ; Copia sobre el buffer de pantalla
                                              ; el caracter (AL) y el atributo (AH).
;
            ADD      BP,2                     ; Suma 2 al contador del buffer de
                                              ; pantalla. (porque cada caracter
                                              ; va acompañado de un atributo).
;
            JMP      ETIQUETA1                ; Vuelve a repetir el proceso
;
ETIQUETA2:                                    ; Etiqueta de la sentencia siguiente.
            RET                               ; Recupera de la pila la dirección
                                              ; de terminación.
;
NOMBRE1     ENDP                              ; Fin del único procedimiento.
;----------------------
CODIGO      ENDS                              ; Fin del segmento CODIGO
            END                               ; Fin del programa
```

PROGRAMAS

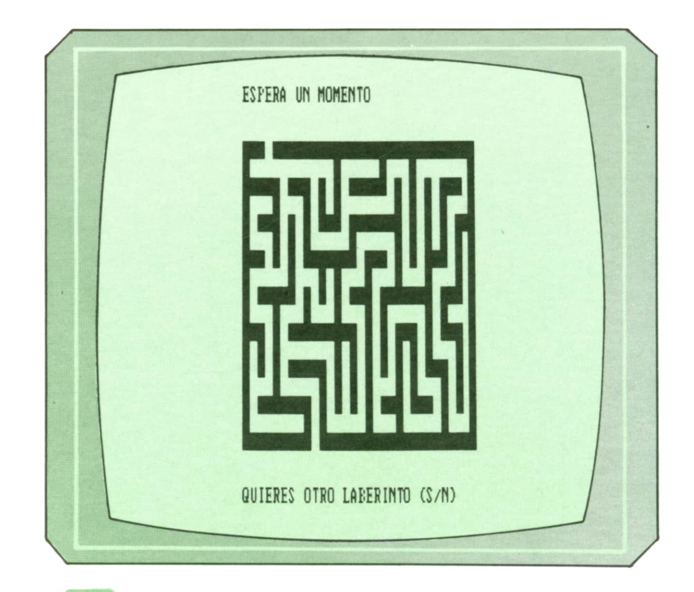
EDUCATIVOS • DE UTILIDAD • DE GESTION • DE JUEGOS



Programa: Diseñador de laberintos

STE programa nos permitirá crear nuestros propios laberintos de forma que los podamos utilizar en cualquiera de los programas que nosotros hagamos. Lo único que hay que darle al programa es la anchura y la altura que queramos que tenga el laberinto. el resto lo hará el ordenador. Todos los laberintos tendrán solución. Esto es, todos los laberintos tendrán una entrada y una salida.

El programa se realizó en un IBM pc, pero puede ser ejecutado en cualquier ordenador que disponga de GWBASIC.



Ejemplo de un laberinto realizado por el programa «diseñador de laberintos».

Para los ordenadores AMSTRAD, COMMODORE y MSX se proponen las siguientes modificaciones.

```
1000 REM **************
1010 REM * LABERINTOS *
1020 REM **************
1030 REM
1040 REM (c) EDICIONES SIGLO CULTURAL
1050 REM (c) 1987
1060 REM
1070 CLS
1080 PRINT " L A B E R I N T O S"
1090 PRINT "---------------------"
1100 PRINT:PRINT
1110 PRINT " Este programa nos permitira"
1120 PRINT "crear laberintos de distintas"
1130 PRINT "dimensiones y siempre con"
1140 PRINT "solucion."
1150 PRINT
1160 PRINT " Para ello, solo tenemos que"
1170 PRINT "dar el ancho y el largo que"
```

```
1180 PRINT "queremos que tenga el laberinto."
1190 PRINT
1200 PRINT "PULSA UNA TECLA PARA EJEMPLOS"
1210 LET A$=INKEY$
1220 IF A$="" THEN GOTO 1210
1230 CLS
1240 PRINT "LABERINTO DE 3x3"
1250 PRINT "----------------"
1260 PRINT:PRINT:PRINT
1270 LET L=3
1280 LET A=3
1290 GOSUB 2000
1300 GOSUB 4000
1310 PRINT
1320 PRINT "PULSA UNA TECLA PARA VER MAS"
1330 LET A$=INKEY$
1340 IF A$="" THEN GOTO 1330
1350 CLS
1360 PRINT "LABERINTO DE 10x6"
1370 PRINT "-----------------"
1380 PRINT:PRINT:PRINT
1390 CLEAR
1400 LET L=10
1410 LET A=6
1420 GOSUB 2060
1430 GOSUB 4000
1440 PRINT
1450 PRINT "PULSA UNA TECLA"
1460 LET A$=INKEY$
1470 IF A$="" THEN GOTO 1460
1480 CLS
1490 PRINT "Dime el tamano de un laberinto"
1500 CLEAR
1510 PRINT:PRINT
1520 INPUT "ANCHO = ";L
1530 PRINT:PRINT
1540 INPUT "LARGO = ";A
1550 CLS
1560 PRINT "ESPERA UN MOMENTO"
1570 PRINT:PRINT
1580 GOSUB 2000
1590 GOSUB 4000
1600 PRINT:PRINT
1610 PRINT "QUIERES OTRO LABERINTO (S/N)
1620 LET A$=INKEY$
1630 IF A$="" THEN GOTO 1620
1640 IF A$="S" OR A$="s" THEN GOTO 1480
1650 IF A$="N" OR A$="n" THEN CLS:END
1660 GOTO 1620
2000 REM
2010 REM **********************
2020 REM * PROGRAMA PRINCIPAL *
2030 REM **********************
2040 REM
2050 RANDOMIZE TIMER
2060 DIM A(L,A):DIM B(L,A)
2070 LET Q=0
2080 LET Z=0
2090 LET M=INT(RND*L+1):LET X=M
2100 LET C=1
2110 LET A(X,1)=C
2120 LET C=C+1
2130 LET R=X
2140 LET S=1
2150 GOTO 2260
2160 IF R<>L THEN GOTO 2240
2170 IF S<>A THEN GOTO 2210
2180 LET R=1
2190 LET S=1
```

```
2200 GOTO 2250
2210 LET R=1
2220 LET S=S+1
2230 GOTO 2250
2240 LET R=R+1
2250 IF A(R,S)=0 THEN GOTO 2160
2260 IF R-1=0 THEN GOTO 2620
2270 IF A(R-1,S)<>0 THEN GOTO 2620
2280 IF S-1=0 THEN GOTO 2430
2290 IF A(R,S-1)<>0 THEN GOTO 2430
2300 IF R=L THEN GOTO 2340
2310 IF A(R+1,S)<>0 THEN GOTO 2340
2320 LET X=INT(RND*3+1)
2330 ON X GOTO 3000,3070,3140
2340 IF S<>A THEN GOTO 2380
2350 IF Z=1 THEN GOTO 2410
2360 LET C=1
2370 GOTO 2390
2380 IF A(R,S+1)<>0 THEN GOTO 2410
2390 LET X=INT(RND*3+1)
2400 ON X GOTO 3000,3070,3230
2410 LET X=INT(RND*2+1)
2420 ON X GOTO 3000,3070
2430 IF R=L THEN GOTO 2540
2440 IF A(R+1,S)<>0 THEN GOTO 2540
2450 IF S<>A THEN GOTO 2490
2460 IF Z=1 THEN GOTO 2520
2470 LET Q=1
2480 GOTO 2500
2490 IF A(R,S+1)<>0 THEN GOTO 2520
2500 LET X=INT(RND*3+1)
2510 ON X GOTO 3000,3140,3230
2520 LET X=INT(RND*2+1)
2530 ON X GOTO 3000,3140
2540 IF S<>A THEN GOTO 2580
2550 IF Z=1 THEN GOTO 2610
2560 LET Q=1
2570 GOTO 2590
2580 IF A(R,S+1)<>0 THEN GOTO 2610
2590 LET X=INT(RND*2+1)
2600 ON X GOTO 3000,3230
2610 GOTO 3000
2620 IF S-1=0 THEN GOTO 2830
2630 IF A(R,S-1)<>0 THEN GOTO 2830
2640 IF R=L THEN GOTO 2750
2650 IF A(R+1,S)<>0 THEN GOTO 2750
2660 IF S<>A THEN GOTO 2700
2670 IF Z=1 THEN GOTO 2730
2680 LET Q=1
2690 GOTO 2710
2700 IF A(R,S+1)<>0 THEN GOTO 2730
2710 LET X=INT(RND*3+1)
2720 ON X GOTO 3070,3140,3230
2730 LET X=INT(RND*2+1)
2740 ON X GOTO 3070,3140
2750 IF S<>A THEN GOTO 2790
2760 IF Z=1 THEN GOTO 2820
2770 LET Q=1
2780 GOTO 2800
2790 IF A(R,S+1)<>0 THEN GOTO 2820
2800 LET X=INT(RND*2+1)
2810 ON X GOTO 3070,3230
2820 GOTO 3070
2830 IF R=L THEN GOTO 2930
2840 IF A(R+1,S)<>0 THEN GOTO 2930
2850 IF S<>A THEN GOTO 2890
2860 IF Z=1 THEN GOTO 2920
2870 LET Q=1
2880 GOTO 3080
```

```
2890 IF A(R,S+1)<>0 THEN GOTO 2920
2900 LET X=INT(RND*2+1)
2910 ON X GOTO 3140,3230
2920 GOTO 3140
2930 IF S<>A THEN GOTO 2970
2940 IF Z=1 THEN GOTO 2990
2950 LET Q=1
2960 GOTO 2980
2970 IF A(R,S+1)<>0 THEN GOTO 2990
2980 GOTO 3230
2990 GOTO 3430
3000 LET A(R-1,S)=C
3010 LET C=C+1
3020 LET B(R-1,S)=2
3030 LET R=R-1
3040 IF C=L*A+1 THEN RETURN
3050 LET Q=0
3060 GOTO 2260
3070 LET A(R,S-1)=C
3080 LET C=C+1
3090 LET B(R,S-1)=1
3100 LET S=S-1
3110 IF C=L*A+1 THEN RETURN
3120 LET Q=0
3130 GOTO 2260
3140 LET A(R+1,S)=C
3150 LET C=C+1
3160 IF B(R,S)=0 THEN GOTO 3190
3170 LET B(R,S)=3
3180 GOTO 3200
3190 LET B(R,S)=2
3200 LET R=R+1
3210 IF C=L*A+1 THEN RETURN
3220 GOTO 2620
3230 IF Q=1 THEN GOTO 3330
3240 LET A(R,S+1)=C
3250 LET C=C+1
3260 IF B(R,S)=0 THEN GOTO 3290
3270 LET B(R,S)=3
3280 GOTO 3300
3290 LET B(R,S)=1
3300 LET S=S+1
3310 IF C=L*A+1 THEN RETURN
3320 GOTO 2260
3330 LET Z=1
3340 IF B(R,S)=0 THEN GOTO 3380
3350 LET B(R,S)=3
3360 LET Q=0
3370 GOTO 3430
3380 LET B(R,S)=1
3390 LET Q=0
3400 LET R=1
3410 LET S=1
3420 GOTO 2250
3430 GOTO 2160
4000 REM
4010 REM ***************************
4020 REM * IMPRESION DEL LABERINTO *
4030 REM ***************************
4040 REM
4050 FOR I=1 TO L
4060    IF I=M THEN GOTO 4090
4070    PRINT CHR$(219);CHR$(219);
4080    GOTO 4100
4090    PRINT CHR$(219);" ";
4100 NEXT I
4110 PRINT CHR$(219)
4120 FOR J=1 TO A
4130    PRINT CHR$(219);
```

```
4140     FOR I=1 TO L
4150        IF B(I,J)<2 THEN GOTO 4180
4160        PRINT "   ";
4170        GOTO 4190
4180        PRINT " ";CHR$(219);
4190     NEXT I
4200     PRINT
4210     FOR I=1 TO L
4220        IF B(I,J)=0 THEN GOTO 4260
4230        IF B(I,J)=2 THEN GOTO 4260
4240        PRINT CHR$(219);" ";
4250        GOTO 4270
4260        PRINT CHR$(219);CHR$(219);
4270     NEXT I
4280     PRINT CHR$(219)
4290 NEXT J
4300 RETURN
```

COMMODORE:

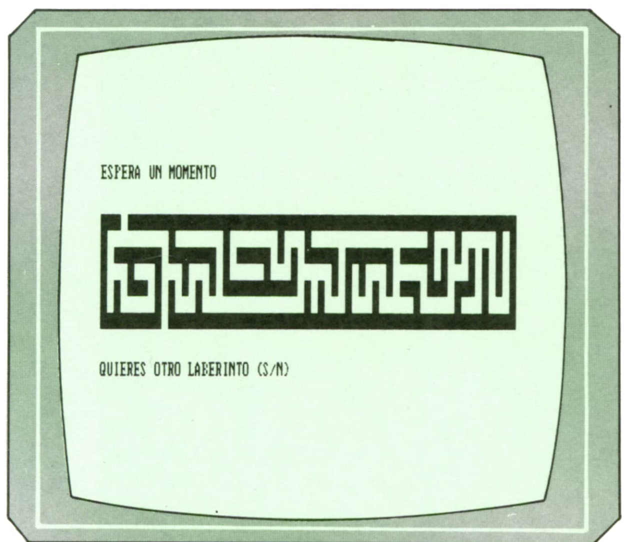
```
1070 PRINT CHR$ (147)
1210 GET A$
1230 PRINT CHR$ (147)
1330 GET A$
1350 PRINT CHR$ (147)
1460 GET A$
1480 PRINT CHR$ (147)
1550 PRINT CHR$ (147)
1620 GET A$
2050 LET F=RND (-TI)
2090 LET M=INT (RND (1)*L+1): LET X=M
2320 LET X=INT (RND (1)*3+1)
2390 LET X=INT (RND (1)*3+1)
2410 LET X=INT (RND (1)*3+1)
2500 LET X=INT (RND (1)*3+1)
2520 LET X=INT (RND (1)*2+1)
2590 LET X=INT (RND (1)*2+1)
2710 LET X=INT (RND (1)*3+1)
2730 LET X=INT (RND (1)*2+1)
2800 LET X=INT (RND (1)*2+1)
2900 LET X=INT (RND (1)*2+1)
4070 PRINT CHR$ (166); CHR$ (166);
4090 PRINT CHR$ (166); " ";
4110 PRINT CHR$ (166)
4130 PRINT CHR$ (166);
4180 PRINT " "; CHR$ (166);
4240 PRINT CHR$ (166); CHR$ (166);
4280 PRINT CHR$ (166)
```

AMSTRAD:

```
2050 RANDOMIZE TIME
2090 LET M=INT (RND (1)*L+1): LET X=M
2320 LET X=INT (RND (1)*3+1)
2390 LET X=INT (RND (1)*3+1)
2410 LET X=INT (RND (1)*2+1)
2500 LET X=INT (RND (1)*3+1)
2520 LET X=INT (RND (1)*2+1)
2590 LET X=INT (RND (1)*2+1)
2710 LET X=INT (RND (1)*3+1)
2730 LET X=INT (RND (1)*2+1)
2800 LET X=INT (RND (1)*2+1)
2900 LET X=INT (RND (1)*2+1)
4070 PRINT CHR$ (143); CHR$ (143);
4090 PRINT CHR$ (143); " ";
4110 PRINT CHR$ (143)
4130 PRINT CHR$ (143);
4180 PRINT CHR$ " "; CHR$ (143);
4240 PRINT CHR$ (143); CHR$ (143);
4280 PRINT CHR$ (143);
```

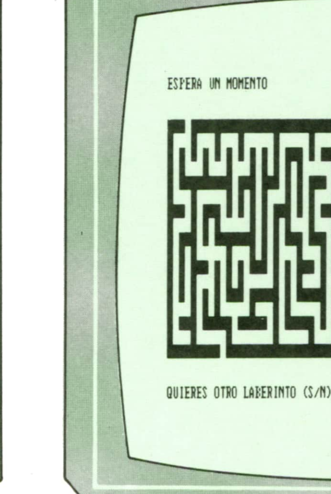
MSX:

```
2050 LET F=RND (-TIME)
2090 LET M=INT (RND (1)*L+1): LET X=M
2320 LET X=INT (RND (1)*3+1)
2390 LET X=INT (RND (1)*3+1)
2410 LET X=INT (RND (1)*2+1)
2500 LET X=INT (RND (1)*3+1)
2520 LET X=INT (RND (1)*2+1)
2590 LET X=INT (RND (1)*2+1)
2710 LET X=INT (RND (1)*3+1)
2730 LET X=INT (RND (1)*2+1)
2800 LET X=INT (RND (1)*2+1)
2900 LET X=INT (RND (1)*2+1)
```





Un laberinto más largo que alto.





Un super laberinto.

Para que el programa nos cree un laberinto sólo tenemos que asignarle a la variable L el valor de la longitud que queremos que tenga el laberinto (ANCHO) y a la variable A el valor de la altura que queremos que tenga. Una vez hecho esto, hacemos un GOSUB a la línea 2000 para que nos lo cree. Si queremos imprimir el laberinto sólo tenemos que hacer GOSUB 4000 una vez que éste esté creado.

Hay que tener cuidado de no darle unos valores muy altos de L y A para que el laberinto no se salga de la pantalla.

Programa Baby Fruits

El siguiente programa, válido únicamente para el SPECTRUM, nos permitirá jugar al conocido juego de las máquinas de los bares que dan premios en metálico. Este tipo de juegos es conocido como BABY FRUITS. No hace falta explicar nada sobre el mismo, ya que todos conocemos perfectamente su funcionamiento.

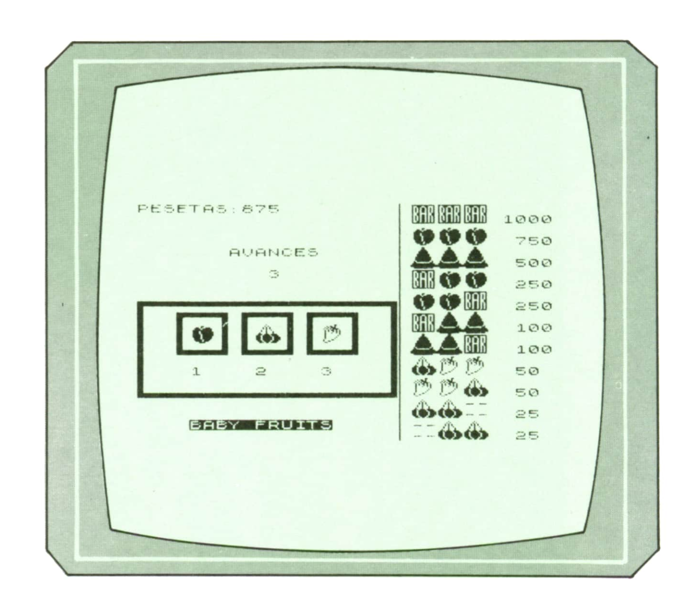
```
   1 DEF FN R()=1+INT (RND*5)
   2 DEF FN P$(A$,Y,X)=CHR$ 22+CHR$ Y+CHR$ X+A$( TO 4)+CHR$ 22+CHR$ (Y+1)+CHR$ X
+A$(5 TO )
  10 INK 7: PAPER 0: BORDER 0: CLS
  20 GO SUB 1290
  30 REM DIBUJO DE LA PANTALLA
  40 PLOT 161,0
  50 DRAW 0,175
  60 BRIGHT 1
  70 PRINT AT 0,21;"QRQRQR"
  80 PRINT AT 1,21;"STSTST 1000"
  90 PRINT AT 2,21; INK 5;"IJIJIJ"
 100 PRINT AT 3,21; INK 5;"KLKLKL"; INK 7;"  750"
 110 PRINT AT 4,21; INK 6;"MNMNMN"
 120 PRINT AT 5,21; INK 6;"OPOPOP"; INK 7;"  500"
 130 PRINT AT 6,21;"QR"; INK 5;"IJIJ"
 140 PRINT AT 7,21;"ST"; INK 5;"KLKL"; INK 7;"  250"
 150 PRINT AT 8,21; INK 5;"IJIJ"; INK 7;"QR"
 160 PRINT AT 9,21; INK 5;"KLKL"; INK 7;"ST  250"
 170 PRINT AT 10,21;"QR"; INK 6;"MNMN"
 180 PRINT AT 11,21;"ST"; INK 6;"OPOP"; INK 7;"  100"
 190 PRINT AT 12,21; INK 6;"MNMN"; INK 7;"QR"
 200 PRINT AT 13,21; INK 6;"OPOP"; INK 7;"ST  100"
```

```
210 PRINT AT 14,21; INK 2;"EF"; INK 7;"ABAB"
220 PRINT AT 15,21; INK 2;"GH"; INK 7;"CDCD  50"
230 PRINT AT 16,21;"ABAB"; INK 2;"EF"
240 PRINT AT 17,21;"CDCD"; INK 2;"GH"; INK 7;"  50"
250 PRINT AT 18,21; INK 2;"EFEF"; INK 7;"--"
260 PRINT AT 19,21; INK 2;"GHGH"; INK 7;"--  25"
270 PRINT AT 20,21;"--"; INK 2;"EFEF"
280 PRINT AT 21,21;"--"; INK 2;"GHGH"; INK 7;"  25"
290 BRIGHT 0
300 PRINT AT 18,2; PAPER 2;"                    "
310 PRINT AT 19,2; PAPER 2;"                    "
320 PRINT AT 20,2; INK 7; PAPER 2;"  BABY FRUITS    "
330 PRINT AT 21,2; PAPER 2;"                    "
340 PRINT AT 0,0;"PESETAS:1000"
350 INK 1: PAPER 7
360 PRINT AT 17,0;"13333333333333333332"
370 FOR G=16 TO 10 STEP -1
380 PRINT AT G,0;"5                  5"
390 NEXT G
400 PRINT AT 9,0;"43333333333333333337"
410 PAPER 7: INK 0
420 PRINT AT 10,3;"4337 4337 4337"
430 PRINT AT 11,3;"5QR5 5QR5 5QR5"
440 PRINT AT 12,3;"5ST5 5ST5 5ST5"
450 PRINT AT 13,3;"1332 1332 1332"
460 PAPER 4
470 FOR I=8 TO 3 STEP -1
480 PRINT AT I,3;"              "
490 NEXT I
500 PRINT AT 4,7; INK 0;"AVANCES"
510 PRINT AT 6,10; INK 0;"0"
520 PAPER 7
530 LET PESETAS=1000
540 REM BUCLE PRINCIPAL DE LA JUGADA
550 PRINT #1; INVERSE 1;"UNA TECLA PARA JUGAR"
560 PRINT AT 0,0; PAPER 0; INK 7;"PESETAS:";PESETAS;"  "
570 PAUSE 0
580 LET PESETAS=PESETAS-25
590 PRINT AT 0,0; PAPER 0; INK 7;"PESETAS:";PESETAS;"  "
600 INPUT ""
610 DIM C(3)
620 FOR I=1 TO 3
630 LET C(I)=1+INT (RND*5)
640 NEXT I
650 LET L=1+INT (RND*5)
660 FOR I=1 TO L
670 PRINT FN P$(F$(FN R()),11,4): BEEP .1,-12
680 PRINT FN P$(F$(FN R()),11,9): BEEP .1,-12
690 PRINT FN P$(F$(FN R()),11,14): BEEP .1,-12
700 NEXT I
710 PRINT FN P$(F$(C(1)),11,4): BEEP .01,24
720 FOR I=1 TO L
730 PRINT FN P$(F$(FN R()),11,9): BEEP .1,-12
740 PRINT FN P$(F$(FN R()),11,14): BEEP .1,-12
750 NEXT I
760 PRINT FN P$(F$(C(2)),11,9): BEEP .01,24
770 FOR I=1 TO L
780 PRINT FN P$(F$(FN R()),11,14): BEEP .1,-12
790 NEXT I
800 PRINT FN P$(F$(C(3)),11,14):: BEEP .01,24
810 GO SUB 850
820 IF PREMIO THEN  GO SUB 970: GO TO 550
830 IF RND>.85 THEN  GO SUB 1060: GO SUB 850: IF PREMIO THEN  GO SUB 970
840 GO TO 550
850 REM CALCULO DEL PREMIO
860 LET PREMIO=0
870 IF C(1)=5 AND C(2)=5 AND C(3)=5 THEN  LET PREMIO=1000: GO TO 960
880 IF C(1)=3 AND C(2)=3 AND C(3)=3 THEN  LET PREMIO=750: GO TO 960
```

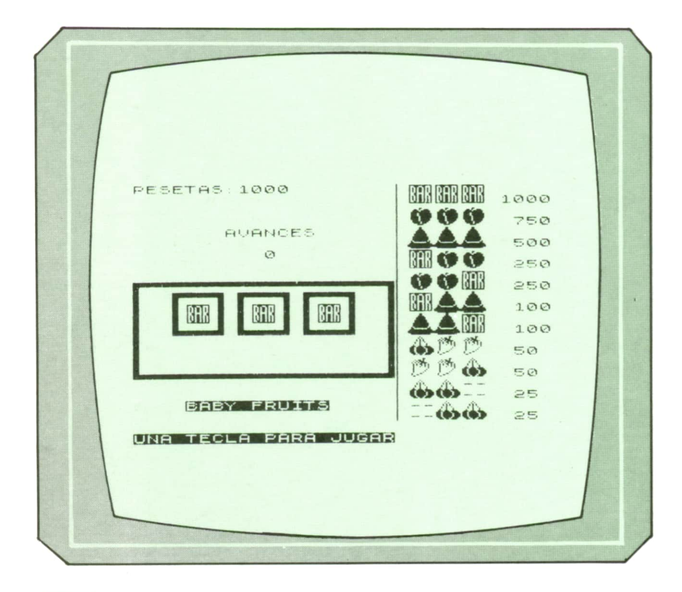
```
 890 IF C(1)=4 AND C(2)=4 AND C(3)=4 THEN  LET PREMIO=500: GO TO 960
 900 IF C(1)=5 AND C(2)=3 AND C(3)=3 THEN  LET PREMIO=250: GO TO 960
 910 IF C(1)=3 AND C(2)=3 AND C(3)=5 THEN  LET PREMIO=250: GO TO 960
 920 IF C(1)=5 AND C(2)=4 AND C(3)=4 THEN  LET PREMIO=100: GO TO 960
 930 IF C(1)=4 AND C(2)=4 AND C(3)=5 THEN  LET PREMIO=100: GO TO 960
 940 IF C(2)=1 AND ((C(1)=2 AND C(3)=1) OR (C(1)=1 AND C(3)=2)) THEN  LET PREMIO
=50: GO TO 960
 950 IF C(2)=2 AND (C(3)=2 OR C(1)=2) THEN  LET PREMIO=25
 960 RETURN
 970 REM PREMIO OBTENIDO
 980 FOR G=1 TO PREMIO/25
 990 FOR I=.1 TO .005 STEP -.01
1000 BEEP I,I*10
1010 NEXT I
1020 LET PESETAS=PESETAS+25
1030 PRINT AT 0,8; PAPER 0; INK 7;PESETAS
1040 NEXT G
1050 RETURN
1060 REM AVANCES
1070 LET NAV=1+INT (RND*4)
1080 PRINT AT 15,4; FLASH 1;"1";AT 15,9;2;AT 15,14;3
1090 PRINT AT 6,10; FLASH 1;NAV
1100 FOR I=NAV TO 1 STEP -1
1110 PRINT AT 6,10; FLASH 1;I
1120 LET CON=0
1130 LET A$=INKEY$
1140 BEEP .01,20: LET CON=CON+1
1150 IF CON>100 THEN  GO TO 1250
1160 IF A$>"3" OR A$<"1" THEN  GO TO 1130
1170 LET T=VAL A$
1180 LET A=4+(T-1)*5
1190 LET C(T)=1+INT (RND*5)
1200 FOR G=1 TO 1+INT (RND*5)
1210 PRINT FN P$(F$(FN R()),11,A)
1220 BEEP .1,-24
1230 NEXT G
1240 PRINT FN P$(F$(C(T)),11,A)
1250 NEXT I
1260 PRINT AT 6,10;0
1270 PRINT AT 15,4;"
1280 RETURN
1290 REM CARGA LOS UDGS
1300 RESTORE 1470
1310 FOR i=USR "a" TO USR "t"+7
1320 READ a
1330 POKE i,a
1340 NEXT i
1350 DIM F$(5,6)
1360 LET ch=144
1370 FOR I=1 TO 5
1380 READ a
1390 LET f$(i, TO 2)=CHR$ 16+CHR$ a
1400 LET f$(i,3 TO 6)=CHR$ ch+CHR$ (ch+1)+CHR$ (CH+2)+CHR$ (CH+3)
1430 LET ch=ch+4
1440 NEXT i
1450 RETURN
1460 REM  DATA DE LOS UDG
1470 DATA 0,0,11,21,42,21,20,17
1480 DATA 0,96,64,192,240,168,168,40
1490 DATA 16,16,16,16,33,38,24,0
1500 DATA 8,16,32,64,128,0,0,0
1510 DATA 0,1,2,5,5,9,9,17
1520 DATA 0,0,128,64,64,32,16,16
1530 DATA 59,119,119,119,119,59,29,0
1540 DATA 188,222,239,239,239,222,188,0
1550 DATA 0,0,1,13,30,63,61,61
1560 DATA 0,0,128,56,252,252,254,254
1570 DATA 62,62,62,30,31,15,3,0
1580 DATA 254,254,252,252,120,112,192,0
```

```
1590 DATA 1,1,3,7,7,15,15,31
1600 DATA 128,128,192,224,224,240,240,248
1610 DATA 31,31,47,48,127,255,255,63
1620 DATA 248,248,244,12,254,255,255,252
1630 DATA 0,127,78,85,85,85,85,76
1640 DATA 0,254,98,170,170,170,170,38
1650 DATA 77,85,85,85,85,77,127,0
1660 DATA 166,170,170,170,170,170,254,0
1670 REM  DATA PARA LA MATRIZ DE LAS FRUTAS
1680 DATA 2,2,5,6,0
```

Sólo hay que decir que las letras que aparecen subrayadas en el listado hay que introducirlas con el cursor en modo GRAPHICS. Para poner el cursor en este modo, tenemos que pulsar a la vez, las teclas CAPS-SHIFT y 9. En ese momento el cursor se volverá como una G parpadeante. Para volver al modo normal del cursor, tenemos que pulsar de nuevo CAPTS-SHIFT y 9.



El juego nos da avances de vez en cuando.

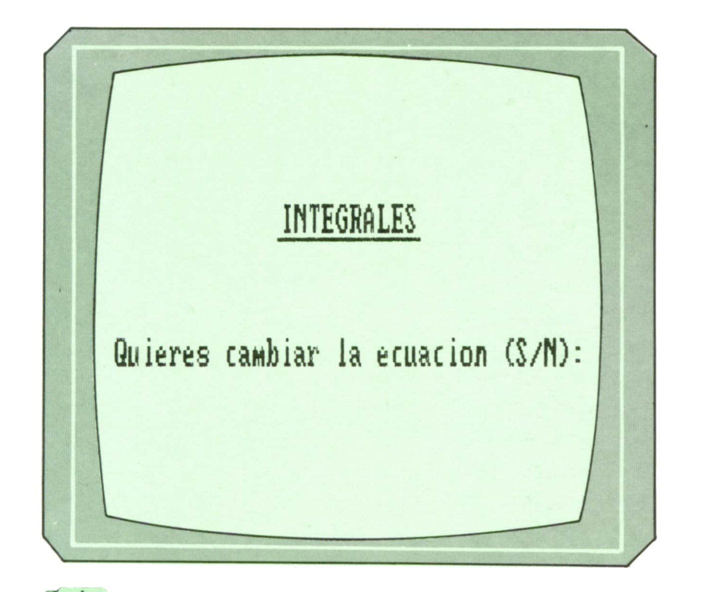


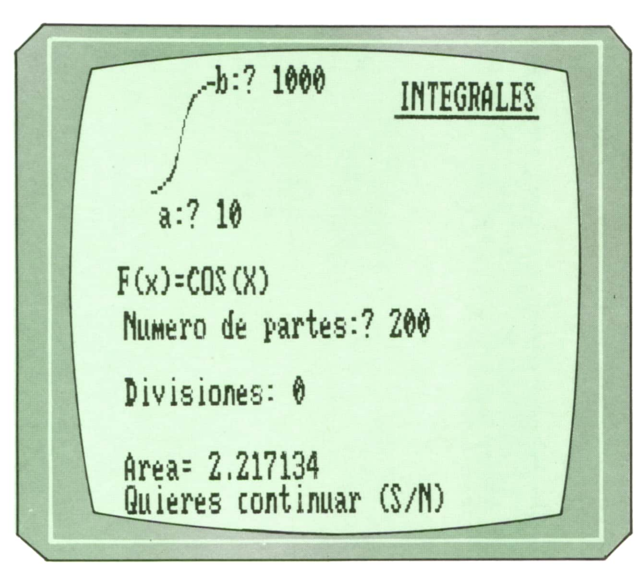

El programa en plena ejecución.

Programa: Integrales para IBM

Este programa, muy apropiado para estudiantes a partir de tercero de BUP, nos va a permitir el cálculo de cualquier integral definida entre dos valores cualesquiera.

El programa puede utilizarse lo mismo como programa autónomo que como subrutina de un programa que realice el usuario.






El programa nos permite cambiar la función a integrar.

Aspecto de la pantalla una vez registrados los cálculos para saber la integral definida entre 10 y 1000 de la función cos (x).

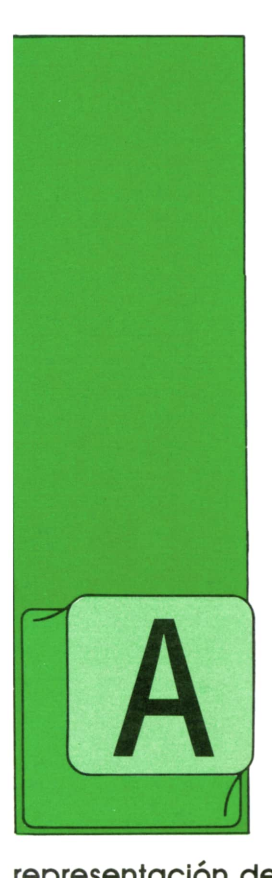
Esta es la versión para IBM y puede funcionar en todos aquellos ordenadores que dispongan del GWBASIC (o BASICA) y que tengan alguna tarjeta gráfica.

```
100 REM ***********************************************************************
110 REM *****       PROGRAMA PARA CALCULAR INTEGRALES POR PARTES        ******
120 REM *****       VALIDO PARA IBM CON TARJETA DE GRAFICOS             ******
130 REM ***********************************************************************
140 REM
150 REM ***********************************************************************
160 REM ***********        POR :  CARLOS A. MARIA MORIN       *****************
170 REM ***********************************************************************
180 REM
190 REM ***********************************************************************
200 REM *********      (C.) EDICIONES SIGLO CULTURAL, 1987      ***************
210 REM ***********************************************************************
220 REM
230 CLS
240 KEY OFF
250 SCREEN 2
260 LOCATE 1,35
270 PRINT"INTEGRALES"
280 KEY 1,CHR$(13)+"GOTO 440"+CHR$(13)
290 SCREEN 2
300 LINE (270,8)-STEP(82,0)
310 CIRCLE (100,20),20,1,1.8,3
320 LINE (79,20)-STEP(-5,10)
330 CIRCLE (53,30),20,1,5,0
340 LOCATE 2,13
350 PRINT"b:"
360 LOCATE 6,9
370 PRINT"a:"
380 LOCATE 5,23
390 PRINT"Quieres cambiar la ecuacion (S/N):"
400 LET A$=INKEY$
410 IF A$="S" OR A$="s" THEN 850
420 IF A$="N" OR A$="n" THEN GOTO 470
430 GOTO 400
440 LOCATE 8,1
450 PRINT "     F(x)="
460 PRINT "          "
```

```
470 LOCATE 21,1
480 PRINT SPACE$(240)
490 LOCATE 6,11
500 INPUT A
510 LOCATE 2,15
520 INPUT B
530 LOCATE 15,1
540 INPUT "Numero de partes:";P
550 LET DX=(B-A)/2/P
560 LET D=0
570 LET X=A
580 GOSUB 830
590 LET D=D+Y
600 LET X=X+DX
610 GOSUB 830
620 LET D=(Y*4)+D
630 LET X=X+DX
640 GOSUB 830
650 LET D=Y+D
660 LET P=P-1
670 LOCATE 17,1
680 PRINT"Divisiones:";P
690 IF P<>0 THEN 590
700 LET RE=(D*DX)/3
710 SOUND 425,1
720 LOCATE 20,1
730 PRINT"Area=";RE
740 PRINT"Quieres continuar (S/N)"
750 A$=INKEY$
760 IF ((A$<>"S" AND A$<>"s") AND (A$<>"N" AND A$<>"n")) THEN 750
770 IF A$="S" OR A$="s" THEN 230
780 KEY 1,"LIST"
790 KEY ON
800 CLS
810 PRINT "ADIOS ..."
820 END
830 LET Y=COS(X)
840 RETURN
850 LOCATE 21,23
860 PRINT "CAMBIA LA FORMULA Y, SIN PULSAR RETURN,"
870 LOCATE 23,28
880 PRINT"CUANDO TERMINES PULSA [ F1 ] "
890 LOCATE 5,18
900 PRINT"                                        "
910 LOCATE 8,1
920 EDIT 830
```

TECNICAS DE ANALISIS

ORDINOGRAMAS



ADEMAS de la utilización de los símbolos indicados, conviene sujetarse a unas ciertas normas en la preparación de los organigramas para una mayor claridad en la representación de los procesos:

a) Deberán dibujarse sobre papel normalizado; deben dejarse unos márgenes adecuados (con vistas a su posible reproducción) especialmente en el lado izquierdo, pues en la encuadernación se perdería (en caso contrario) parte de la información.

b) Es básico preparar y rellenar cuidadosamente un cajetín de identificación de todos los procesos. Además, es útil referenciar otros procesos que precedan o vayan a continuación de aquel al que se refiere el organigrama en cuestión.

c) Debe establecerse una norma uniforme para toda la documentación, de modo que exista un organigrama general de la aplicación y otros varios de cada proceso independiente (uno por cada actividad separable del resto). Junto a cada uno de estos organigramas parciales se preparará un ordinograma lógico de cada proceso (más adelante hablaremos de ellos).

d) El flujo lógico de la información irá (salvo excepciones) de la parte superior del dibujo hacia la parte inferior y de izquierda a derecha.

e) En los conectores y las líneas de conexión de símbolos se indicará (con puntas de flecha) el sentido del flujo de la información.

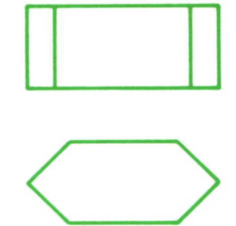
f) Deben identificarse (al menos someramente) los procesos que intervienen en la aplicación y los archivos involucrados.



Ordinogramas

En un segundo nivel de detalle, se suelen preparar unos organigramas de detalle o microorganigramas, para cada proceso separado dentro de la aplicación. En estos organigramas de detalle, es importante resaltar los procesos lógicos que se producen y el flujo de los datos. Actividades como selección, clasificación, etc., son las importantes a reseñar en cada documento (más que la relación de unos procesos con otros o la estructura general de las actividades que vendrán reflejadas en los organigramas de la aplicación); por esta razón a estos organigramas se les denomina, en ocasiones, ordinogramas de proceso.

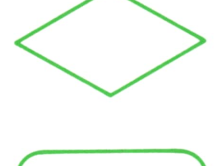
Para la preparación de los ordinogramas se utilizan la mayoría de los símbolos usados en los organigramas, y algunos otros más concretos que señalan el flujo de los datos y los procesos básicos a desarrollar con ellos:



Subrutina o proceso predefinido.

Símbolo de preparación. Para reseñar las operaciones adicionales necesarias para algún proceso especial.

Bifurcación. Se utiliza como un conector (del que parten y salen lí-



neas de flujo de datos) pero incluyendo algún criterio de toma de decisión.

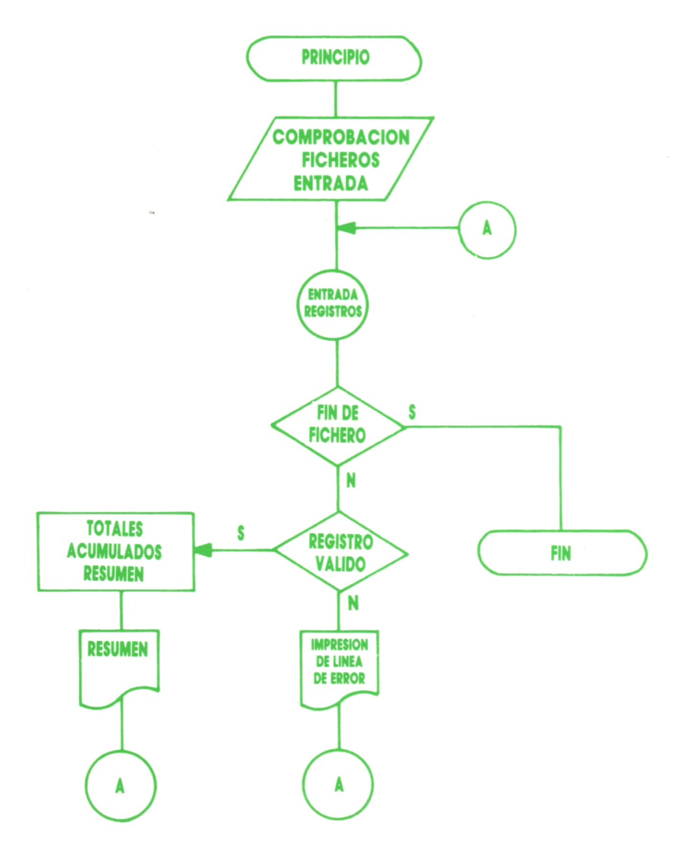


Símbolo terminal. Marca el comienzo o final de un proceso.

La utilización básica de estos símbolos es equivalente, pero el concepto del diagrama y su finalidad varían. En cualquier caso, el criterio fundamental que se debe aplicar es que, dentro del conjunto de normas establecidas, los organigramas y ordinogramas sean de la máxima claridad posible.

Deberán tenerse en cuenta, en la confección de los ordinogramas, las siguientes normas:

a) establecer un ordinograma para cada proceso separable en que se puede dividir la aplicación, dando indicaciones (con el símbolo de relación correspondiente) para las correspondencias entre los diferentes procesos.



b) Aunque en el correspondiente símbolo de tratamiento o clasificación se indiquen los parámetros a utilizar, es conveniente reseñar a nivel de presentación del ordinograma las funciones básicas y parámetros que caracterizan cada proceso.

c) Las líneas de flujo deben ser verticales y horizontales (no oblicuas). Las líneas que parten de cada archivo deben ir a un solo símbolo de proceso. Si deben llegar a un símbolo de archivo varias conexiones (de procesos diferentes) o si han de salir de un archivo varias líneas deberán incluirse los oportunos conectores (círculos de enlace) de donde bifurquen estas líneas.

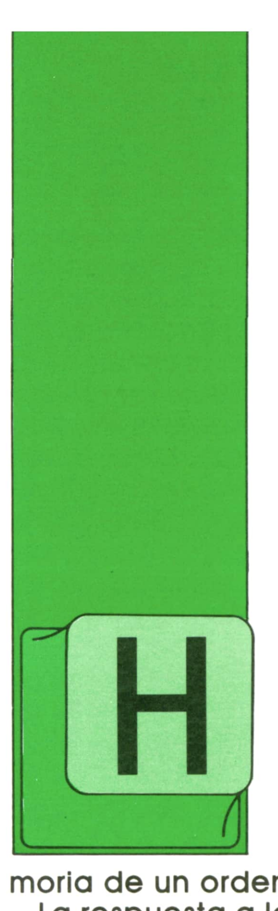
d) Para mayor claridad, se suelen poner los símbolos de salida impresa debajo del tratamiento en el que se prepara dicho impreso. Los símbolos de entrada o salida magnética (cintas y discos) se deben poner a izquierda o derecha respectivamente del símbolo del tratamiento correspondiente. Si un archivo es de acceso doble (para entrada y para salida) se debe poner a la izquierda del símbolo de tratamiento, preferentemente, y unido a éste por una doble flecha para indicar que se produce un flujo bidireccional de datos.

e) Se supone que el flujo lógico de presentación de archivos y procesos es de izquierda a derecha y de arriba abajo. Cuando el flujo de datos va según esta norma, no es necesario utilizar puntas de flecha: si los datos no van según esas direcciones deben utilizarse dichos símbolos.

f) Deben evitarse las intersecciones de líneas de conexión y, en caso imprescindible, limitarlas al máximo. Si el resultado incluye muchos cruces y se pierde claridad, separar una parte del proceso llevándolo a otro ordinograma o, dentro de la misma hoja, haciendo otro ordinograma. En ambos casos es imprescindible establecer claramente los puntos de relación de las diferentes partes, mediante los correspondientes símbolos conectores.

TECNICAS DE PROGRAMACION

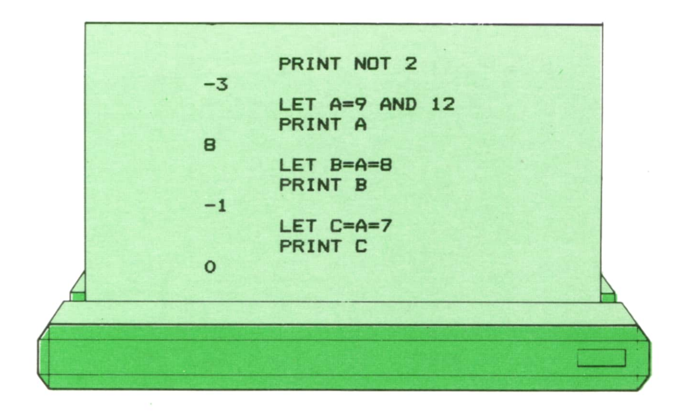
EXPRESIONES



HEMOS visto que las expresiones de resultado lógico pueden tomar sólo dos valores diferentes: «verdadero» y «falso». ¿Cómo se representan estos dos valores en la memoria de un ordenador?

La respuesta a la pregunta anterior depende del lenguaje de programación. En general, se utilizan ciertas combinaciones de bits para representarlos. Por ejemplo, en BASIC, el valor «falso» se representa generalmente con un cero, mientras cualquier valor distinto de cero significa «verdadero». En particular, las expresiones de comparación (como X=3) devuelven un valor igual a cero si la condición no se cumple, y un valor igual a −1 si la condición se cumple. Naturalmente, estos valores también estarán representados internamente como números enteros binarios de cierto número de dígitos. Los números negativos se representan, como ya hemos mencionado, con la notación de «complemento a dos».

En BASIC es posible asignarle a una variable el resultado de una operación lógica e imprimir su valor. Además, las operaciones lógicas (como AND) pueden actuar sobre valores no lógicos (enteros). En este caso, la operación se realiza bit a bit, como si el número entero, expresado en el sistema binario, estuviera en realidad formado por un vector de ceros y unos, donde los ceros significan «falso» y los unos «verdadero». Veamos algunos ejemplos:



El primer caso (NOT 2, que da un resultado de −3) precisa para interpretarlo conocer la notación de complemento a dos que representa los números negativos. En efecto, la representación binaria en 16 bits del número entero 2, es 0000000000000010. La operación NOT, aplicada bit a bit, cambia todos los ceros (falso) en unos (verdadero) y todos los unos (verdadero) en ceros (falso). Por tanto, NOT 2 sería igual a 1111111111111101, que es la representación en complemento a dos del número negativo −3.

El segundo caso es más fácil: se trata de realizar la operación AND entre los dos enteros 9 y 12. La representación binaria del 9 es 0000000000001001 y la del 12 es 0000000000001100. La operación AND, realizada bit a bit, convierte dos unos en un uno (la conjunción es verdadera si los dos operandos lo son) y cualquier otra combinación de bits en cero (la conjunción es falsa en cuanto uno de los operandos lo sea). Por tanto, la operación se realizará así:

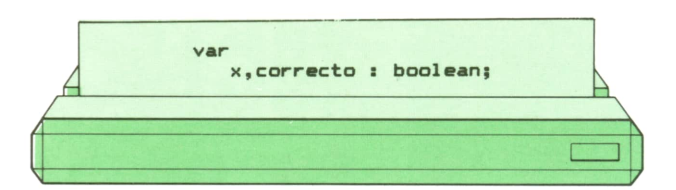
```
        0000000000001001
AND
        0000000000001100
da
        0000000000001000
```

Es decir, el resultado será igual a 0000000000001000 en binario, que es el número 8. Este valor ha sido asignado por la instrucción (LET A=9 AND 12) a la variable A, que por tanto valdrá precisamente 8.

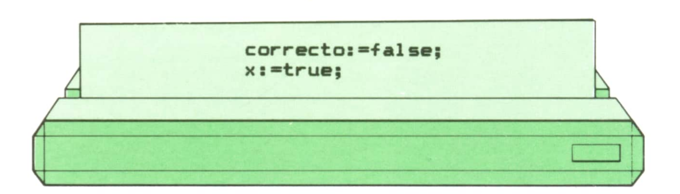
Veamos el tercer ejemplo. La instrucción (LET B=A=8) es muy significativa, pues nos presenta claramente otra de las anomalías del lenguaje BASIC, especialmente visible en intérpretes que permiten prescindir de la palabra reservada LET en las asignaciones de valor, pues entonces la instrucción anterior queda reducida a (B=A=8). Esta anomalía consiste en el hecho de que, en BASIC, el mismo símbolo (=) representa dos operaciones completamente diferentes: la asignación de valor y la comparación de dos valores. El intérprete decide cuál es el significado apropiado en cada caso en función del contexto. Una asignación de valor debe estar situada a la izquierda de una instrucción, por lo que el primer signo igual de la instrucción anterior es interpretado de esta manera. En cambio, si existe algo a la izquierda del signo igual y de sus operandos, el intérprete llega a la conclusión de que se trata de una comparación. Por lo tanto, la instrucción (B=A=8) significa: «Comparar el valor de la variable A con 8. Si son iguales, asignarle −1 ("verdadero") a la variable B. Si A no vale 8, asignarle 0 ("falso") a la variable B». Como, en nuestro caso, A valía 8, el valor de B pasa a ser −1, como podemos comprobar en la instrucción siguiente.

En cambio, en el cuarto ejemplo, la instrucción (LET C=A=7) asigna a C el valor cero («falso»), pues no es verdad que el valor de A (8) sea igual a 7.

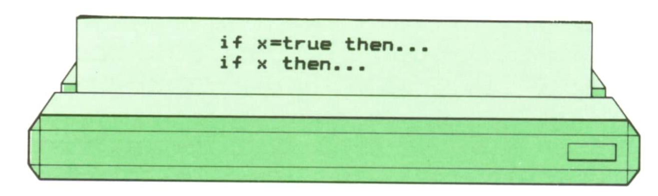
En BASIC no existen variables de tipo lógico. Si le asignamos un valor lógico a una variable (como en los ejemplos anteriores) el valor asignado pasa a ser, inmediatamente, entero. En PASCAL, sin embargo, sí se pueden definir variables de tipo lógico, lo que se consigue con una declaración parecida a la siguiente:



que define las dos variables «x» y «correcto», cuyo valor será lógico (booleano, en la terminología de PASCAL). Una variable booleana sólo puede adoptar uno de los dos valores: «verdadero» (representado por la palabra inglesa «TRUE») y «falso» (representado por la palabra inglesa «FALSE»). Estas dos palabras están reservadas para este uso, por lo que se pueden utilizar directamente en asignaciones a variables lógicas:

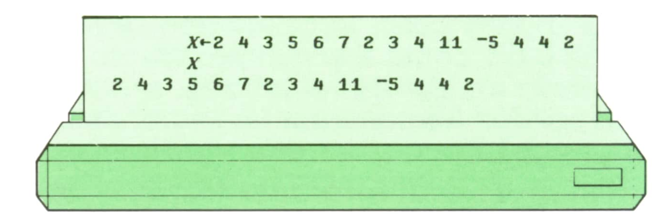


También se pueden utilizar como términos de comparación en una expresión relacional:

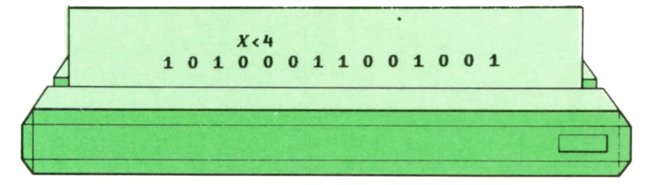


Las dos líneas de este último ejemplo son totalmente equivalentes. Pues (x=true) es verdad («true») sí y sólo si el valor de x es «true». Por lo tanto, la expresión (x=true) puede sustituirse simplemente por (x).

En APL, la verdad o la falsedad de una operación se representan siempre con los valores cero («falso») y uno («verdadero»). Este cero y este uno pueden considerarse, indistintamente, como valores lógicos o como valores enteros, con los que se puede operar. Esto es útil a menudo. Por ejemplo, supongamos que tenemos una variable X cuyo valor es una serie de números enteros:

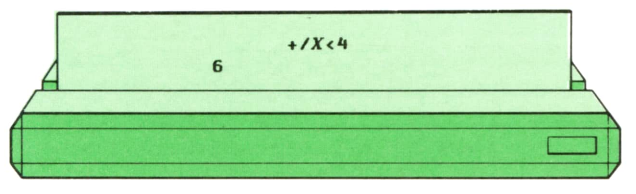


Supongamos que nos interesara saber cuáles de los valores de X son menores que 4. Bastaría con realizar la siguiente comparación:



donde los unos del resultado corresponden a los elementos de X para los que la relación es «verdadera» (es decir, para los que son menores que 4), mientras los ceros corresponden a aquellos elementos para los que la relación es «falsa» (es decir, para los que son mayores o iguales que 4).

Pues bien: supongamos que lo que realmente deseamos saber no es cuáles son los elementos de X que son menores que 4, sino cuántos son. Su número, sin importarnos su orden ni su posición. Podríamos obtenerlo con la expresión siguiente:



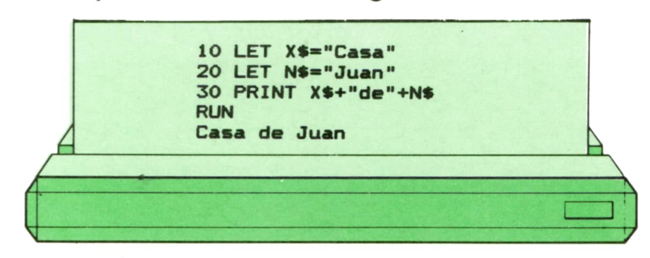
En efecto, el par de símbolos (+/) obtiene en APL la suma de todos los elementos de la serie de números a la que se aplica. En nuestro caso, como el resultado de la relación lógica puede considerarse también como una serie de números enteros (ceros y unos), dicha suma tiene que coincidir con el número de unos, que no es otra cosa que el número de los elementos de X que eran menores que 4.

Existen otras muchas aplicaciones de las operaciones y los tipos de datos lógicos, pero ya tenemos bastante por el momento.

Expresiones literales

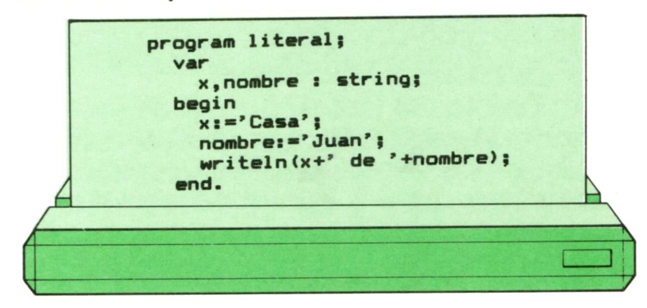
Llamaremos expresiones literales a las que se aplican a datos literales (cadenas de caracteres) para producir resultados literales.

La operación literal más frecuente en los lenguajes de programación es la concatenación, que une dos cadenas de caracteres entre sí para formar una cadena nueva, cuya longitud es la suma de las longitudes de las dos cadenas a las que se aplica. En BASIC esta operación se representa con el signo +:



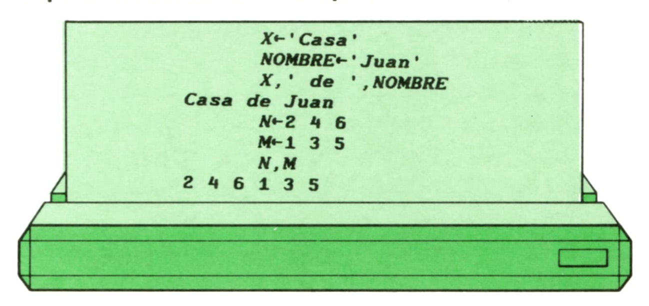
Obsérvese que, al concatenar dos cadenas de caracteres BASIC, se introduce automáticamente un espacio en blanco entre ellas.

En los compiladores de PASCAL que reconocen el tipo «string» existe también la operación de concatenación, que por analogía con BASIC se suele representar también con el signo +. Sin embargo, en este caso no se introducen automáticamente espacios en blanco entre los caracteres que se concatenan.



El programa anterior produce el mismo resultado que el ejemplo BASIC equivalente. Obsérvese que, en este caso, hemos tenido que incluir explícitamente los blancos de separación a ambos lados de la palabra «de».

En APL, la operación de concatenación se representa con una coma y se aplica por igual a datos literales o numéricos. Tampoco en este caso se añaden automáticamente espacios en blanco para separar los literales que se concatenan.

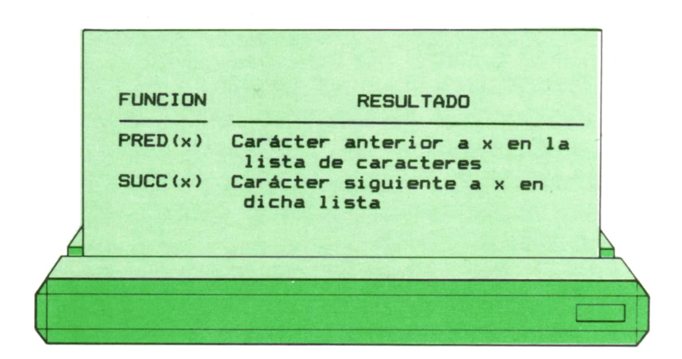


Además de la operación de concatenación, existen en los lenguajes de programación otras funciones u operaciones que actúan sobre datos literales. Veamos una lista de algunas funciones literales BASIC:

FUNCION	RESULTADO
LEFT$(x$,n)	Los n caracteres a la izquierda de x$
MID$(x$,n,m)	m caracteres de x$ a partir del que hace el número n
RIGHT$(x$,n)	Los n caracteres a la derecha de x$
SPACE$(n)	Una cadena de n espacios en blanco
STRING$(n,x$)	El primer carácter de x$ repetido n veces

En PASCAL, dado que el tipo «string» no es uno de los tipos estándar de este lenguaje, las funciones que actúan sobre él dependen del compilador. Una de las más corrientes es la función COPY, que actúa de manera equivalente a la función MID$ de BASIC.

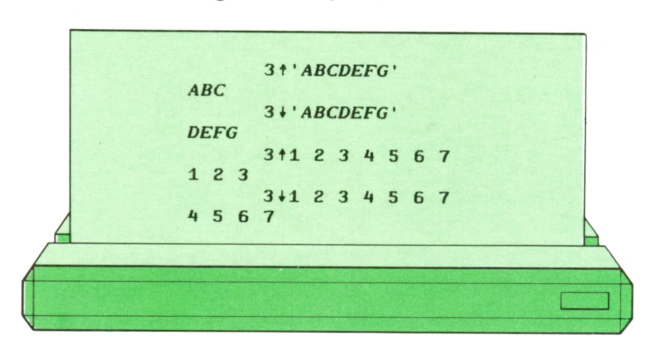
En cambio, el tipo «char» sí es un tipo estándar de PASCAL, por lo que las dos funciones siguientes aparecerán normalmente en todos los compiladores:



FUNCION	RESULTADO
PRED(x)	Carácter anterior a x en la lista de caracteres
SUCC(x)	Carácter siguiente a x en dicha lista

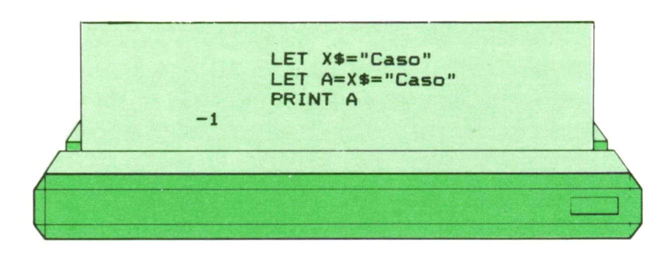
En APL existen también funciones que pueden actuar por igual sobre datos numéricos o literales. Por ejemplo, la operación representada por una flecha ascendente extrae de la serie de datos situadas a su derecha, cualquiera que sea su tipo, tantos elementos como indique el número situado a su izquierda. De igual manera, la operación representada por una flecha descendente elimina de la serie de datos situada a su derecha tantos elementos como indique el número situado a su izquierda.

Veamos algunos ejemplos:

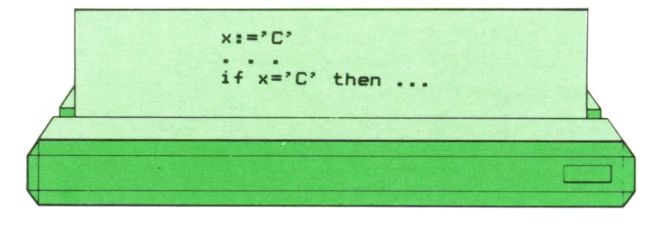


Expresiones mixtas

Estrictamente hablando, son las que mezclan datos de distintos tipos o que producen un resultado de tipo diferente al de sus operandos. Por ejemplo: números con literales, datos numéricos con datos lógicos, etc. Ya hemos visto algunos casos. Las operaciones de comparación, por ejemplo (como X=2) se utilizan para comparar dos números, obteniendo el resultado «verdadero» o «falso». Estas mismas operaciones se pueden utilizar con datos de caracteres. Por ejemplo, en BASIC:



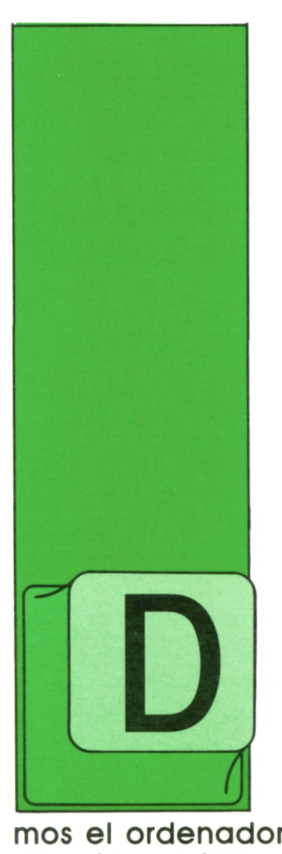
En PASCAL:



En APL:



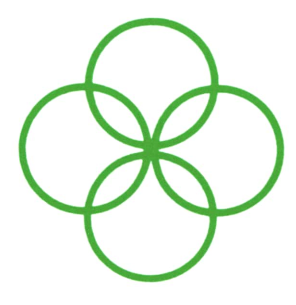
LOGO



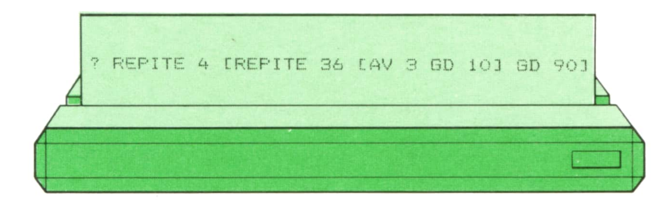
Utilización de los procedimientos

A sabemos que podemos enseñar a la tortuga a hacer cosas nuevas definiendo un procedimiento y que no lo olvidará hasta que se lo digamos o hasta que apaguemos el ordenador. Por tanto, la palabra que da nombre a ese procedimiento es como si fuera un comando más para la tortuga. Por ello, la podemos utilizar, por ejemplo, dentro de un REPITE.

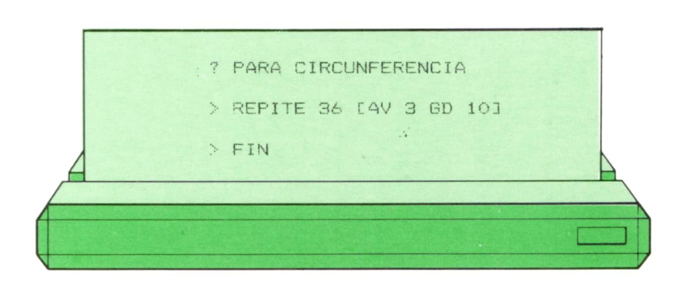
Supongamos que queremos pintar una figura formada por 4 circunferencias:



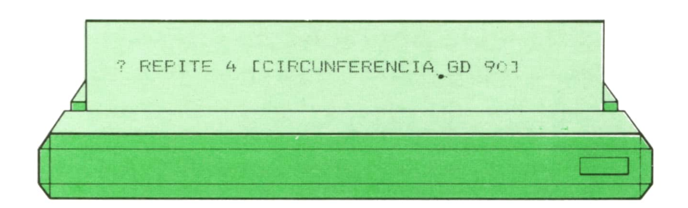
Una manera de hacerlo ya la conocemos:



La otra consiste en definir un procedimiento para que la tortuga aprenda a dibujar una circunferencia:

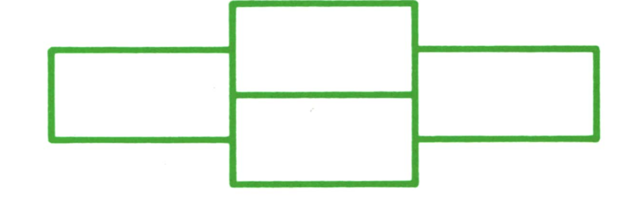


y ponerlo dentro del REPITE:

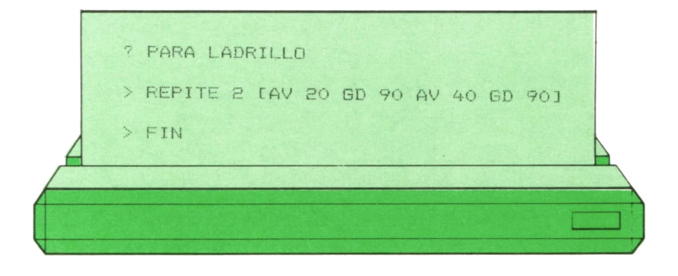


Estas nuevas palabras también se pueden usar junto con otros comandos, no sólo con el REPITE. Veámoslo con un ejemplo.

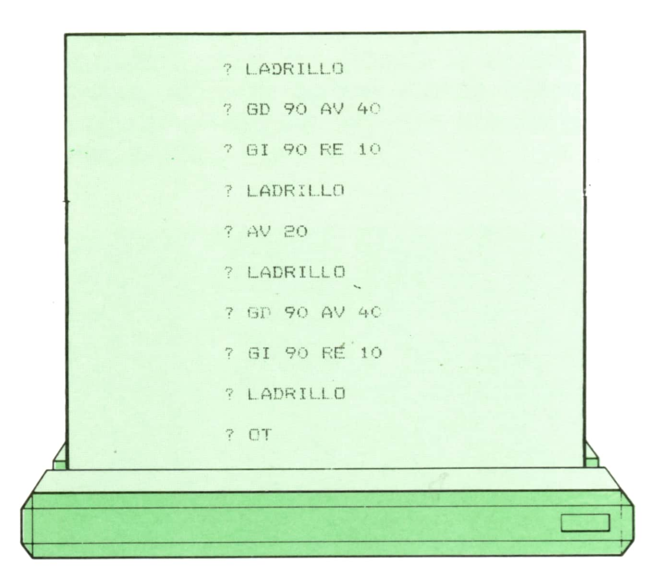
Para obtener un dibujo formado por un conjunto de ladrillos



primero hemos de enseñarle a la tortuga a hacer un ladrillo. Por ello, definimos un procedimiento:



y ahora escribimos los siguientes comandos:



Como vemos, antes de dibujar cada ladrillo hemos de decirle a la tortuga una serie de comandos para que se vaya colocando en la posición que corresponda a ese ladrillo.



Uso de unos procedimientos dentro de otros

Debido a que el nombre de un procedimiento se puede utilizar como si fuera un comando, parece lógico que lo podamos poner dentro de la definición de otro procedimiento.

De esta manera, para realizar un dibujo que sea bastante complicado, en lugar de hacerlo con un solo procedimiento podemos dividirlo en partes más pequeñas y sencillas. Cada una de estas partes se obtiene a partir de su procedimiento correspondiente. Por último, necesitamos definir un procedimiento general que se encargue de usar todos los demás y con el que podamos pintar el dibujo completo.

Por ejemplo, vamos a dibujar una bailarina:

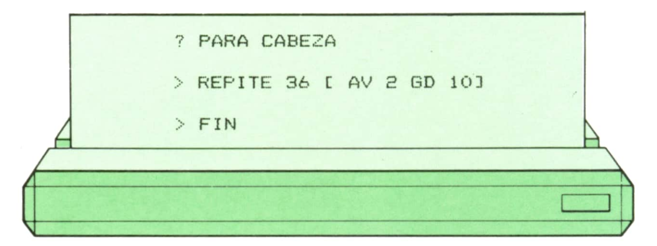


Podemos dividirla en cuatro partes:

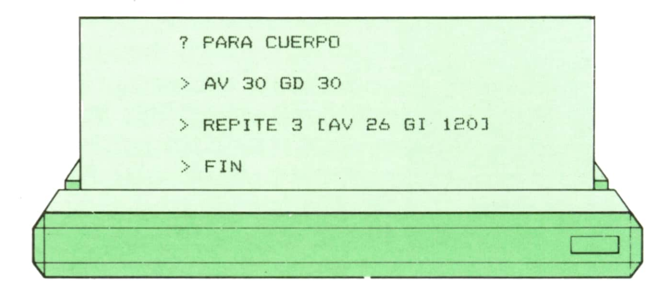
— cabeza
— cuerpo
— brazos
— piernas

Ahora escribimos un procedimiento que se encargue de dibujar cada parte:

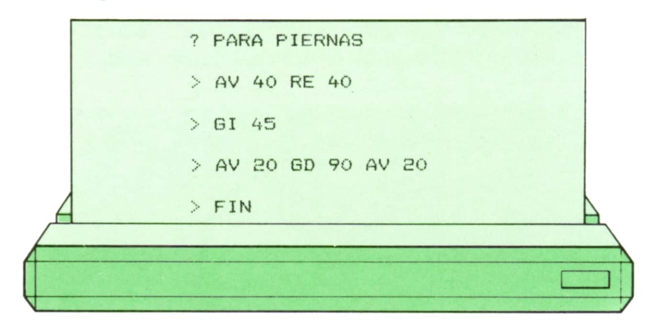
— cabeza



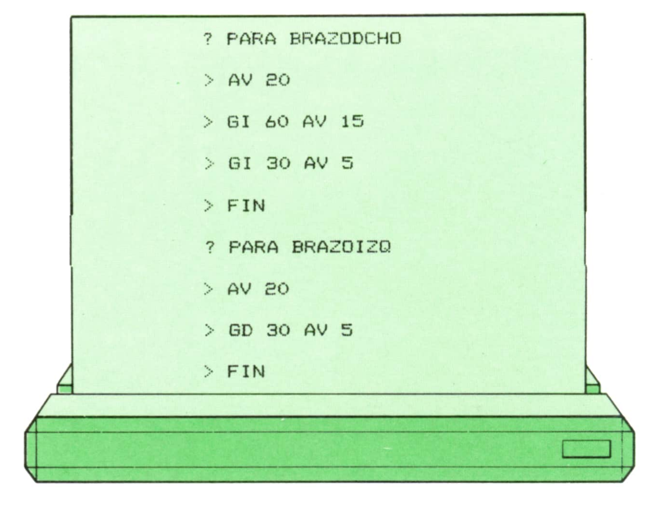
— cuerpo



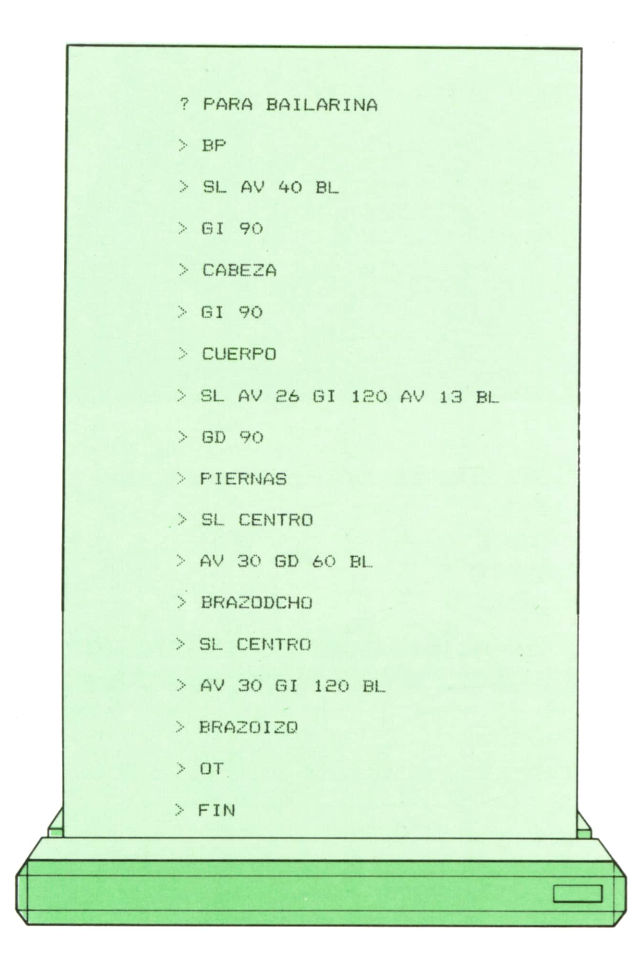
— piernas



— brazos



Nos queda escribir el procedimiento final que tenga como función obtener el dibujo total (la bailarina). Para ello, tenemos que ir colocando a la tortuga en la posición correcta antes de dibujar cada parte:



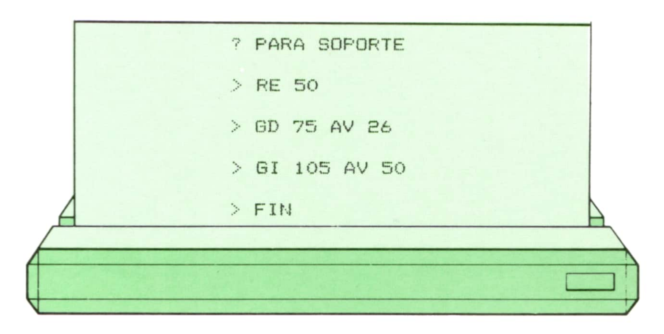
Vamos a hacer lo mismo para dibujar esta noria:

En primer lugar, dividimos la noria en partes más sencillas:

— soporte
— barra
— cabina

En segundo lugar, hemos de enseñarle a la tortuga a hacer cada cosa. Para ello, definimos los procedimientos siguientes:

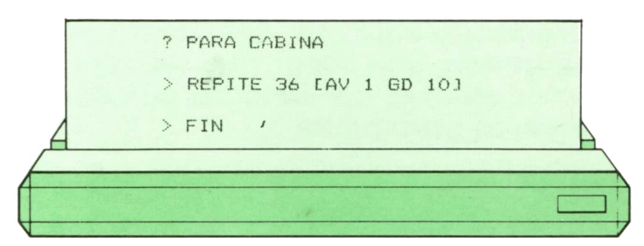
— soporte

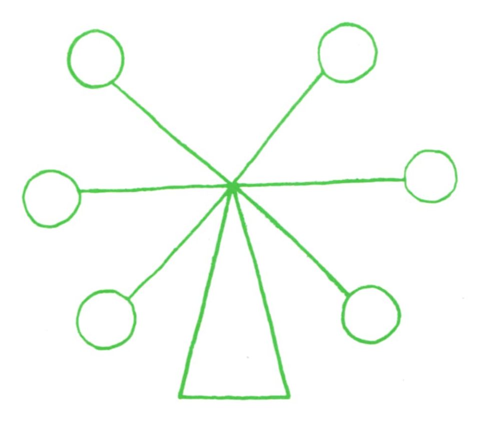


— barra

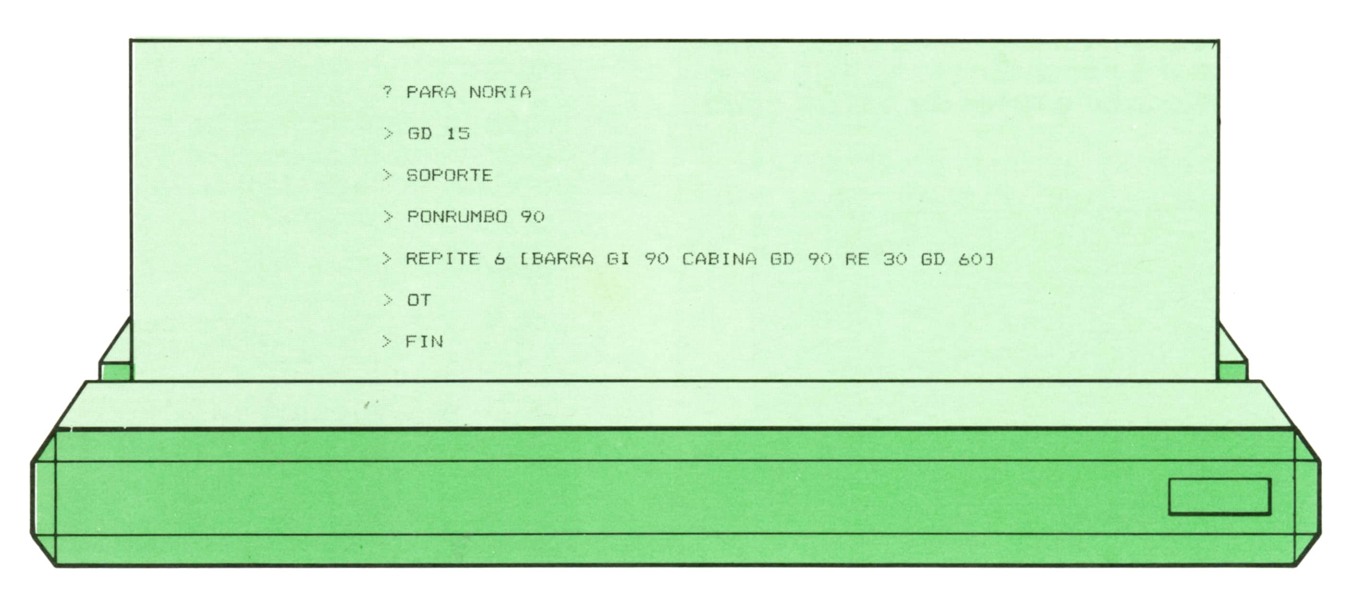


— cabina





Por último, creamos el procedimiento que le enseñe a la tortuga a realizar la noria:



El usar este método de descomponer un dibujo complejo en partes más pequeñas no sólo ayuda a que sea más sencillo de realizar, sino que, además, es mejor a la hora de tener que corregir los procedimientos si no nos sale el dibujo que queremos.

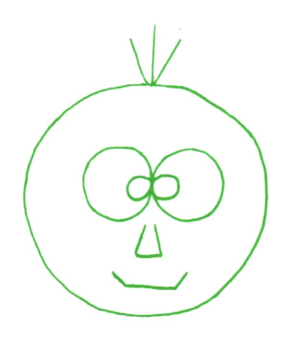
Esto es debido a que sólo necesitaremos cambiar aquel procedimiento correspondiente a la parte del dibujo que esté mal, sin que esto afecte a los que estén bien, ya que no hará falta modificarlos.

Además, si una parte del dibujo se repite varias veces es más cómodo utilizar el procedimiento ya definido que le corresponda que tener que escribir en cada ocasión el conjunto de comandos necesarios para pintarla.



Os proponemos

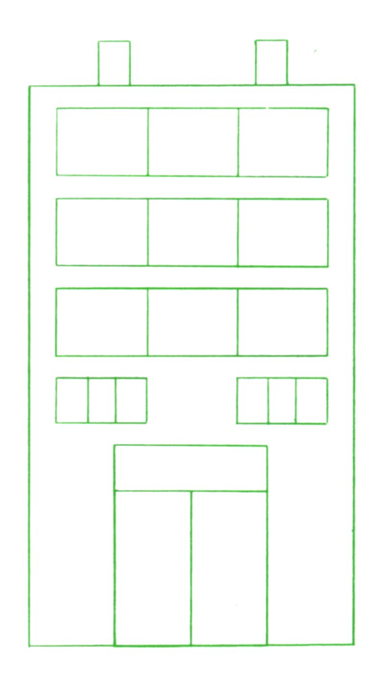
1. Puedes pintar una cara. Te damos como pista las partes en las que la puedes dividir.



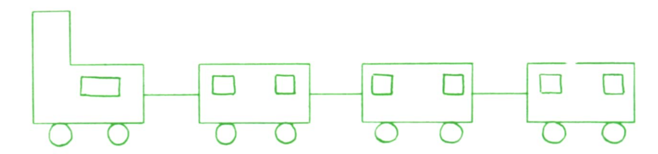
Cara:

— ojos
— nariz
— boca
— pelo

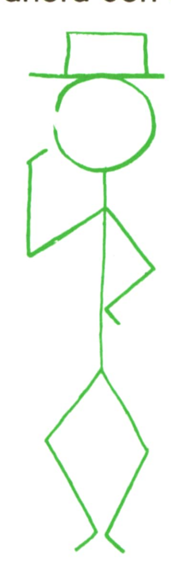
2. También puedes intentar pintar este edificio dividiéndolo en varias partes.



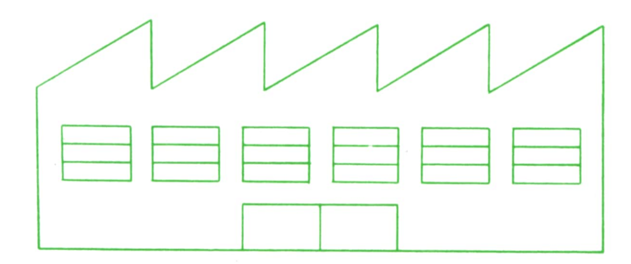
3. ¿Te atreverías a hacer este tren?



4. Prueba ahora con este muñeco.



5. Puedes pintar también una fábrica.



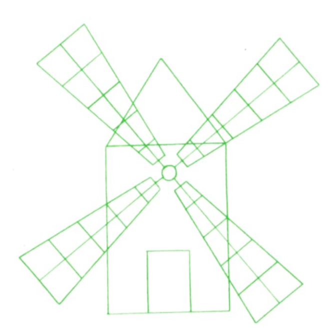
6. Ahora puedes realizar este dibujo:



Para dibujar media circunferencia utiliza el comando REPITE, así:

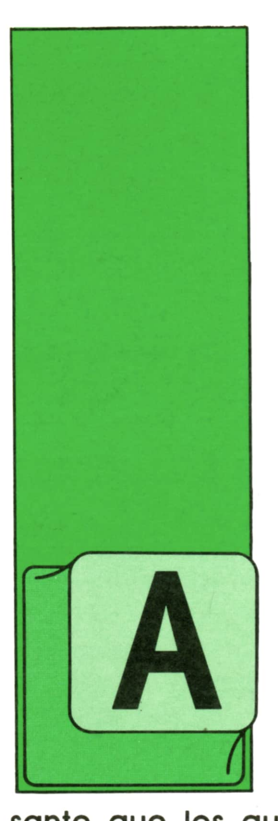
REPITE 18 (AV tamaño GD 10)

7. ¿Qué te parece este molino?



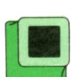
Para dibujar o recorrer la cuarta parte de una circunferencia has de poner:

REPITE 9 (AV tamaño GD 10)



PASCAL

AUNQUE todavía quedan muchos conceptos de PASCAL por estudiar, ya conocemos lo suficiente como para desarrollar algún programa que sea más interesante que los que hemos hecho hasta ahora. Por ello, y para descansar de tanta teoría, vamos a escribir uno que nos permita jugar al «Master Mind» con el ordenador; en principio, seremos sólo nosotros quienes tratemos de adivinar la combinación secreta, aunque en otra ocasión veremos cómo modificar el programa para que también pueda ser el ordenador el que lo intente con una combinación escogida por nosotros.

 El juego del Master Mind

Aunque existen muchas variantes de este juego, vamos a describir una de las más corrientes, que será la que emplearemos; consiste en lo siguiente:

Un jugador (que en nuestro caso será el ordenador) se inventa una combinación secreta formada por cuatro elementos, cada uno de los cuales puede tomar uno cualquiera de entre varios posibles valores. Por ejemplo, los elementos podrían ser cifras comprendidas entre 1 y 8, o letras entre la A y la G, o piezas de 6 posibles colores; de esta manera, podríamos tener combinaciones como 1315 para el primer caso, BAGB para el segundo o «rojo, blanco, rojo, rojo» para el último.

El otro jugador ha de adivinar la combinación; para ello va presentando a su oponente distintas combinaciones, y éste debe informarle sobre lo acertado que sea cada intento de la siguiente manera:

— Por cada uno de los elementos de la nueva combinación que coincida en valor y posición con uno de la combinación secreta, se contabiliza lo que se denomina un «muerto». Por ejemplo, si la combinación a descubrir fuera 1315 tendríamos:

1315

2345 Dos muertos (el 3 y el 5).
3151 Ningún muerto.
5112 Un muerto (el segundo 1).
1315 Cuatro muertos (o sea, ya está).

— Por cada uno de los elementos de la nueva combinación que coincida en valor, pero no en posición, con uno de la combinación secreta, se contabiliza lo que se denomina un «herido», teniendo en cuenta que si un elemento ya ha servido para contabilizar un muerto o un herido, no sirve para contabilizar más de éstos. Por ejemplo:

1315

1222 Ningún herido (el 1 ya dio un muerto).
2164 Un herido (el 1).
3636 Un herido (alguno de los treses).
5112 Dos heridos (el 5 y el primer 1).
3151 Cuatro heridos.

El poseedor de la combinación secreta debe contestar a cada intento del otro jugador con el número de muertos y heridos correspondiente, hasta que se descubra la solución; se trata de hacerlo con el menor número de intentos posible. Suponiendo que las cifras sólo puedan

variar entre 1 y 6, veamos una partida muy sencilla:

6334: da 1 muerto y 0 heridos.

supongamos que el muerto se debe al 4 y probemos a variar las otras cifras:

1224: da 1 muerto y 0 heridos

si fuese el 4, entonces no habría 1, 2, 3 ni 6 y las tres restantes cifras deben ser 4 y/o 5; probaremos, por tanto, lo siguiente:

5544: da 0 muertos y 1 herido

por tanto, no hay ningún 4 y debe haber un 5 en alguna de las posiciones de la derecha, así como 3 ó 6 y 1 ó 2 en las demás; probemos:

6256: da 0 muertos y 1 herido

como hay un 5 seguro, éste está en último lugar y no hay ni 2 ni 6; por la primera combinación sabemos que no puede haber más de un 3, con lo que sólo puede ser 1315 ó 1135.

1315: da 4 muertos

Todo buen programador encontrará claro el proceso deductivo. Una vez descrito el juego, pasemos a escribir el programa.



Creación de la combinación secreta

La combinación, como es lógico, debe crearla el programa al azar. La mayoría de compiladores de PASCAL disponen de alguna función para obtener números al azar que suelen tener nombres del estilo de RANDOM; no obstante, como no es una función estándar, nosotros nos crearemos la nuestra propia.

El sistema que seguiremos es uno muy sencillo que para un problema como el nuestro es más que suficiente; consiste en lo siguiente:

Dado un número aleatorio cualquiera comprendido entre 0 y 1, se obtiene uno nuevo a partir de él tomando la parte fraccionaria del resultado de multiplicarlo por 997.

De esta manera obtenemos, a partir de un número cualquiera (que en nuestro caso se lo daremos nosotros al ordenador) una secuencia de números aleatorios. Si, por ejemplo, quisiéramos obtener números aleatorios enteros comprendidos entre 20 y 29, como los obtenidos están comprendidos entre 0 y 1, bastará con multiplicarlos por 10, truncar a entero el resultado (con lo que tendremos números entre 0 y 9) y sumarles 20.

Vamos a escribir un pequeño programa que presente combinaciones aleatorias de Master Mind; podría ser algo así:

1. Introducir un número cualquiera de arranque.
2. Preguntar el límite superior de los posibles valores.
3. Presentar unas cuantas combinaciones.

Utilizaremos un procedimiento para el punto 1 y otro para obtener números aleatorios. Cada vez que se llame a este último procedimiento, como es necesario utilizar el número anterior, habrá que guardar a éste en una variable global (que llamaremos Aleatorio).

```
program Combinaciones;

 var
  Aleatorio : real;
  Limite,I,J: integer;

(*--------------------------------------------------------*)
procedure ArrancaAzar;
(* Pide el primer número de la serie aleatoria *)

  var Ok: boolean;
 begin
  writeln ('Teclee un número entre 0 y 1, ambos exclusive.');
  repeat
   readln (Aleatorio);
   Ok:= (0.0 < Aleatorio) and (Aleatorio < 1.0);
   if not Ok then writeln ('No vale. Repita.')
  until Ok
 end;

(*--------------------------------------------------------*)
function Azar (Inf,Sup: integer): integer;
```

```
(*   proporciona un número entero aleatorio    *
 *   entre Inf y Sup, ambos inclusive          *)

  var A: real;
 begin
  Aleatorio:= frac (Aleatorio * 997);              (* ¡OJO! *)
  Azar:= Inf + trunc (Aleatorio * (Sup - Inf + 1))
 end;

(*-------------------------------------------------------*)
begin
 ArrancaAzar;
 write ('Números del 1 al...');
 readln (Limite);
 writeln;
 for I:= 1 to 20 do   (* Presentar 20 combinaciones *)
  begin
   for J:=1 to 4 do write (Azar (1, Limite));
   writeln (* salta de línea tras cada combinación *)
  end
end.
```

La función real predefinida Frac, que devuelve la parte fraccionaria de un número, no está disponible con todos los compiladores; si éste es nuestro caso, una solución alternativa sería cambiar la instrucción:

Aleatorio:= frac (Aleatorio * 997);

por:

Aleatorio:= Aleatorio * 997;
Aleatorio:= Aleatorio–trunc (Aleatorio);



Obtención del número de muertos y heridos

Una vez que se ha obtenido una combinación secreta, el programa debe preguntar combinaciones al jugador y devolverle el número de heridos y muertos hasta que éstos sean 4.

Para analizar el número de muertos bastará con comparar la primera cifra de la combinación secreta con la primera de la del jugador, la segunda con la segunda, etc.

Para los heridos, sin embargo, habrá que comparar cada cifra de una combinación con todas las de la otra que ocupen diferente posición. Hay que evitar que cifras que ya han dado lugar a muertos o heridos se vuelvan a contabilizar por haber varias repetidas; por ejemplo, si comparamos por las buenas 1234 con 1122, obtendríamos un muerto y tres heridos, lo cual es erróneo. La solución más fácil es «tachar» cada cifra que haya dado lugar a un muerto o herido cambiándole el valor a otro que sepamos seguro que no va a poder proporcionar más coincidencias; para no perder la combinación original habrá, por tanto, que trabajar con una copia:

```
program  MasterMind;

 var
  Aleatorio    : real;

  S1,S2,S3,S4,       (* Para guardar la clave secreta *)
  C1,C2,C3,C4,       (* Para guardar cada intento     *)
  Muertos,
  Heridos,
  Limite       : integer;
  Letra        : char;

(*-------------------------------------------------------*)
procedure ArrancaAzar;
(* Pide el primer número de la serie aleatoria *)
```

```
    var Ok: boolean;
   begin
    writeln ('Teclee un número entre 0 y 1, ambos exclusive.');
    repeat
      readln (Aleatorio);
      Ok:= (0.0 < Aleatorio) and (Aleatorio < 1.0);
      if not Ok then writeln ('No vale. Repita.')
    until Ok
   end;

 (*----------------------------------------------------------*)
 function Azar (Inf,Sup: integer): integer;
 (*   proporciona un número entero aleatorio    *
  *   entre Inf y Sup, ambos inclusive          *)

   var A: real;
  begin
   Aleatorio:= frac (Aleatorio * 997);               (* ¡OJO! *)
   Azar:= Inf + trunc (Aleatorio * (Sup - Inf + 1))
  end;

 (*----------------------------------------------------------*)
 procedure Analizar;
 (*  Compara S1,S2,S3,S4 con C1,C2,C3,C4 y da     *
  *  el valor adecuado a Muertos y Heridos.       *)

    var
     A1,A2,A3,A4: integer;
   begin
    (* Saca una copia de la clave secreta: *)
    A1:= S1;
    A2:= S2;
    A3:= S3;
    A4:= S4;

    (* Analiza los muertos: *)
    Muertos:= 0;
    if A1 = C1 then begin Muertos:= Muertos + 1; A1:= -1; C1:= -2 end;
    if A2 = C2 then begin Muertos:= Muertos + 1; A2:= -1; C2:= -2 end;
    if A3 = C3 then begin Muertos:= Muertos + 1; A3:= -1; C3:= -2 end;
    if A4 = C4 then begin Muertos:= Muertos + 1; A4:= -1; C4:= -2 end;

    (* Analiza los heridos; las instrucciones marcadas con asteriscos no
       son necesarias pues las variables afectadas no se van a comparar
       más; no obstante, se han dejado para tener una mayor regularidad  *)
    Heridos:= 0;

    if A1 = C2 then begin Heridos:= Heridos + 1; A1:= -1; C2:= -2 end;
    if A1 = C3 then begin Heridos:= Heridos + 1; A1:= -1; C3:= -2 end;
    if A1 = C4 then begin Heridos:= Heridos + 1; A1:= -1; C4:= -2 end;
                                                 (**)

    if A2 = C1 then begin Heridos:= Heridos + 1; A2:= -1; C1:= -2 end;
    if A2 = C3 then begin Heridos:= Heridos + 1; A2:= -1; C3:= -2 end;
    if A2 = C4 then begin Heridos:= Heridos + 1; A2:= -1; C4:= -2 end;
                                                 (**)

    if A3 = C1 then begin Heridos:= Heridos + 1; A3:= -1; C1:= -2 end;
    if A3 = C2 then begin Heridos:= Heridos + 1; A3:= -1; C2:= -2 end;
    if A3 = C4 then begin Heridos:= Heridos + 1; A3:= -1; C4:= -2 end;
                                                 (**)     (**)

    if A4 = C1 then begin Heridos:= Heridos + 1; A4:= -1; C1:= -2 (**) end;
    if A4 = C2 then begin Heridos:= Heridos + 1; A4:= -1; C2:= -2 (**) end;
    if A4 = C3 then begin Heridos:= Heridos + 1; A4:= -1; C3:= -2 (**) end;
                                                 (**)
   end;

 (*----------------------------------------------------------*)
 begin
  ArrancaAzar;
  write ('Números del 1 al...');
  readln (Limite);
  (*---------------------------------------*)
  repeat                  (* repetir partidas *)
    (* Primero, inventarse una clave: *)
    S1:= Azar (1, Limite);
    S2:= Azar (1, Limite);
    S3:= Azar (1, Limite);
    S4:= Azar (1, Limite);

  Aleatorio:= frac (Aleatorio * 997);


  Aleatorio:= Aleatorio * 997;
  Aleatorio:= Aleatorio - trunc (Aleatorio);
```

```
    Obtención del número de muertos y heridos.
    ------------------------------------------
    ClrScr; (* o PAGE, para borrar la pantalla *)
    writeln ('Tras cada cifra, pulse Intro.');
    writeln;
    (*-----------------------------------------*)
    repeat   (* Pedir y analizar combinaciones: *)
      write ('Clave:   ');
      read (C1);
      read (C2);
      read (C3);
      read (C4);

      Analizar;

      writeln ('   m=',Muertos,'   h=',Heridos)
    until Muertos = 4;
    (*-----------------------------------------*)
    writeln;
    write ('¿ Desea otra partida ? (S/N)   ');
    readln (Letra)

  until (Letra = 'N') or (Letra = 'n');
  (*-----------------------------------------*)
  writeln ('Adiós.')

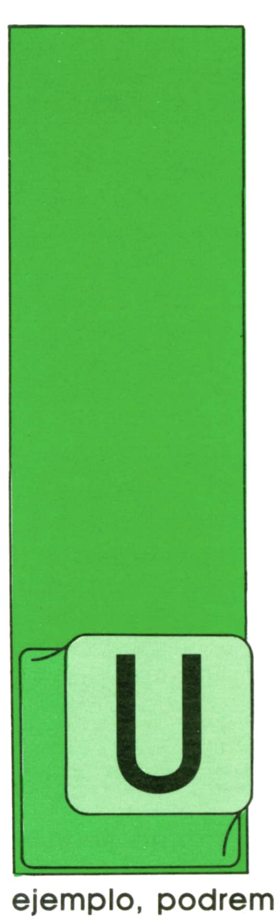
end.
```

Como ejercicio, se podría modificar el programa para que compruebe que los números tecleados son correctos. También se podría mejorar el aspecto estético.

Dentro de poco veremos cómo el analista se puede hacer de una manera mucho más elegante.

OTROS LENGUAJES

ARRAYS Y PUNTEROS

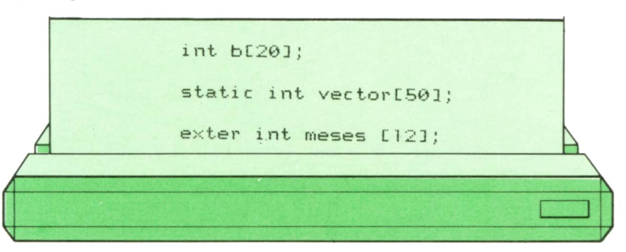




¡Qué es un array?

UN array es una estructura de datos formada por elementos del mismo tipo y utilizados para almacenar una serie de datos que nuestro programa necesita. Por ejemplo, podremos guardar las vocales en un array compuesto por cinco elementos.

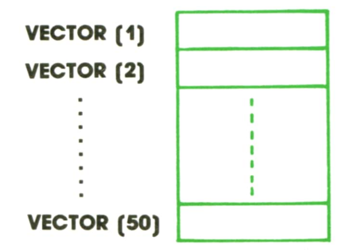
Veamos algunas formas de declarar arrays:



```
int b[20];

static int vector[50];

exter int meses [12];
```



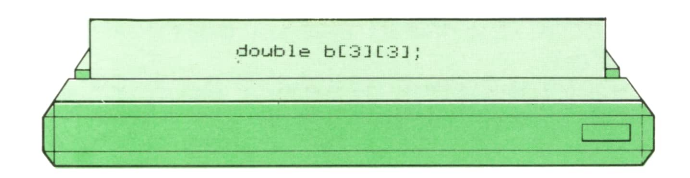


Un array unidimensional de 50 elementos llamado «vector».



Arrays multidimensionales

Los arrays vistos anteriormente se denominan **arrays unidimensionales** o **vectores.** Un array cuyos elementos sean arrays unidimensionales se llama **array bidimensional.** De manera análoga podríamos definir un array tridimensional y así sucesivamente. Para entender mejor este concepto supongamos que definimos el siguiente array:



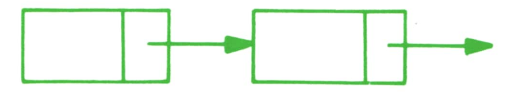
```
double b[3][3];
```

Este array queda definido como una matriz de 3 × 3 elementos en coma flotante y doble precisión.



Punteros

Si en la literatura especializada usted encuentra en alguna ocasión la palabra «pointers», tradúzcala por **puntero,** pues ese es su significado, pero dejando aparte la lingüística, diremos que un **puntero** es un tipo de variable cuyo contenido es una dirección de memoria, donde hay un dato que nos puede ser de interés en un momento dado.





Enlace de datos por medio de punteros.

El operador que nos proporciona un puntero a un variable es el **operador &.**

Por ejemplo, si «y» es una variable, un puntero a dicha variable es &y.

La teoría de **punteros** puede parecer complicada y confusa para los principiantes en este tipo de variables (algo parecido ocurre cuando utilizamos muy a menudo la «proscrita» sentencia «goto»), pero siendo ordenados en nuestros planteamientos, se pueden emplear para conseguir una mayor claridad y simplicidad de nuestros programas.

En C también existen **variables puntero,** es decir, no sólo se asigna a una variable **int** un entero o a una variable **char** un carácter, también a una variable **puntero** se le podrá asignar como valor una dirección.

Por ejemplo:

```
punt = &vect;
```

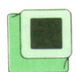
asignará la dirección de **vect** a **punt,** o lo que es igual, se dice que **punt** está apuntando a **vect.**

El **operador indirección** * accede a la variable apuntada por un determinado puntero.

Veamos cómo debemos declarar en un programa los punteros:

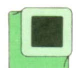
```
int *pa;
float *pf;
```

En las declaraciones anteriores, la primera nos dice que «pa» es un puntero y que «*pa» es de tipo entero, es decir, el valor (*pa) apuntado por «pa» es de tipo entero. Algo análogo sucede con **float *pf,** excepto que es un puntero a una variable **float**.



Punteros y funciones

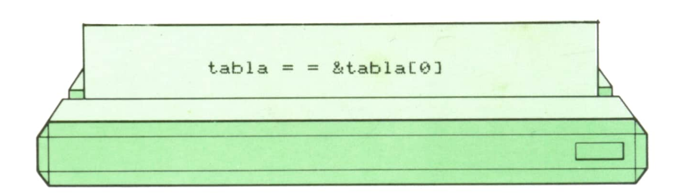
La utilización de punteros para comunicación entre funciones permite alterar las variables de forma parecida a como se modifican los parámetros por «variable» del Pascal.



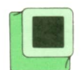
Punteros y arrays

En C existe una estrecha relación entre punteros y arrays lo suficientemente fuerte como para que se les trate simultáneamente. Cualquier tratamiento que podamos hacer con arrays es susceptible de poderse hacer con punteros, pero consiguiendo una mayor rapidez con estos últimos, aunque su comprensión será más difícil.

Supongamos un array llamado «tabla»:

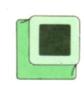


La igualdad anterior se cumple porque tanto «tabla» como &tabla(0) representan la dirección de memoria del primer elemento. Ambos términos no podrán cambiar su valor, pero pueden ser asignados a una variable «puntero» y podemos modificar su valor.



Arrays y funciones

Los arrays, al igual que sucede en Pascal, pueden aparecer como argumentos de funciones. Si intentamos pasar el nombre de un array como argumento de una función, lo que realmente estamos pasando es un puntero, y esta circunstancia es aprovechada por la función para realizar los cambios necesarios en el array.



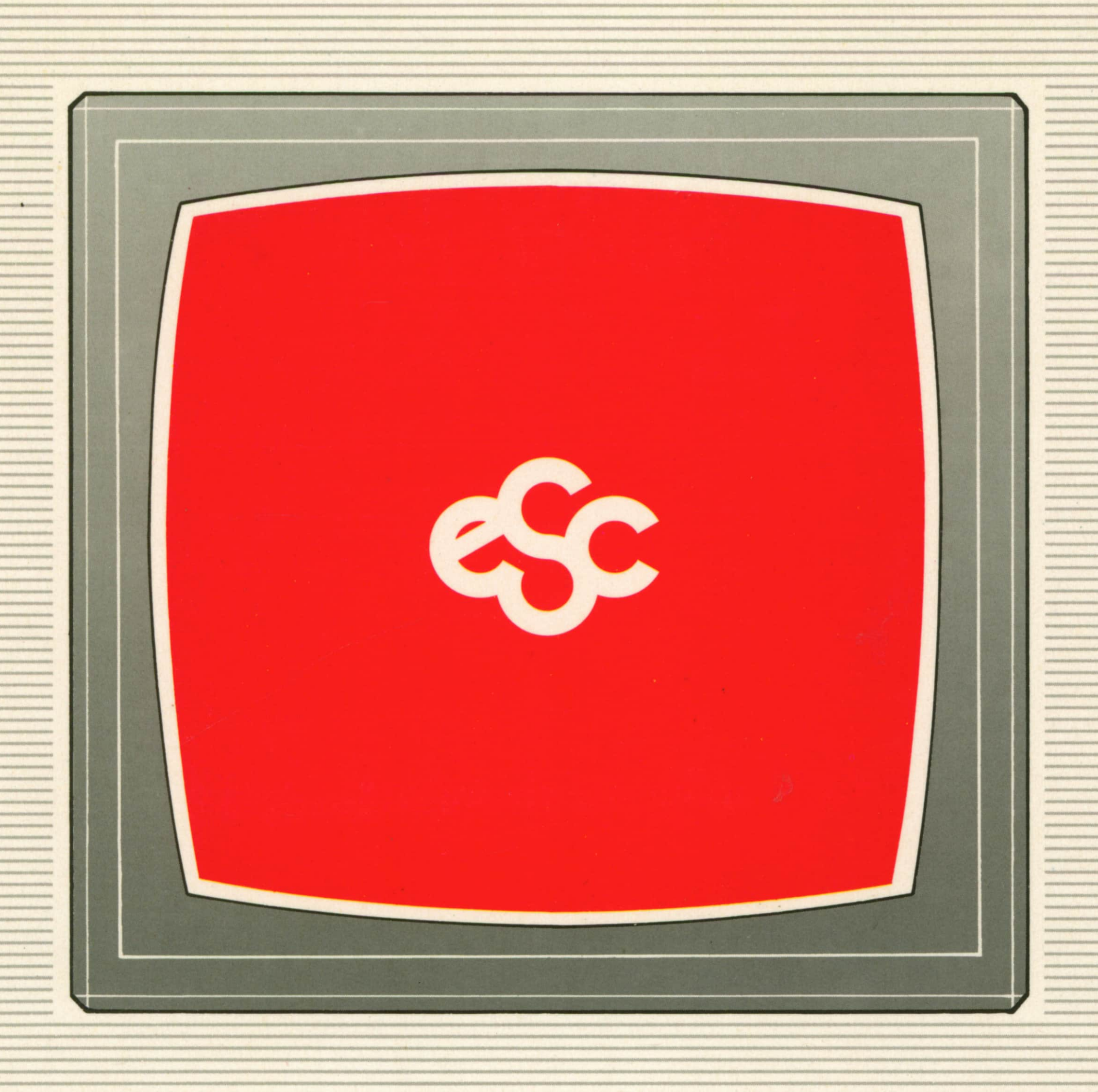
Operaciones básicas con punteros

Con punteros podemos realizar tres operaciones:

— **Asignación:**
Siempre podemos asignar una dirección a un puntero.

— **Incremento:**
Un puntero puede ser incrementado utilizando el operador de incremento. De forma similar podemos decrementarlo.

— **Diferencia de punteros:**
Es posible hallar la diferencia de dos punteros que apuntan al mismo array para averiguar el número de elementos que existen entre ellos.



▼ EDICIONES ▼ SIGLO ▼ CULTURAL ▼